MASTER EN SCIENCES MATHÉMATIQUES À FINALITÉ DIDACTIQUE

Adaptation de la « One Fifth Success Rule » pour le réglage de la taille de la population dans l'algorithme génétique du logiciel MINAMO

Pool Marquez, Sylvério

Award date:
2021

Awarding institution:
Universite de Namur

UNIVERSITE DE NAMUR

Faculté des Sciences

Adaptation de la « One Fifth Success Rule » pour le réglage de la taille de la population dans l'algorithme génétique du logiciel MINAMO

Mémoire présenté pour l'obtention

du grade académique de master en « sciences mathématiques à finalité didactique »

Sylvério Pool Marquez

Juin 2021

UNIVERSITE DE NAMUR

Faculté des Sciences

Adaptation de la « One Fifth Success Rule » pour le réglage de la taille de la population dans l'algorithme génétique du logiciel MINAMO

Promoteurs :

Timoteo Carletti

Annick Sartenaer

Co-promotrice :

Charlotte Beauthier

Mémoire présenté pour l'obtention

du grade académique de master en « sciences mathématiques à finalité didactique »

Sylvério Pool Marquez

Juin 2021

Remerciements

Ce mémoire est la finalité d'un an et demi de travail pour lequel de nombreuses personnes m'ont apporté leur aide ainsi que leur soutien. Tout d'abord, j'aimerais remercier ma co-promotrice C. Beauthier pour l'accompagnement et l'investissement dont elle a fait preuve envers moi tout le long de ce travail. Ensuite, je souhaite adresser mes remerciement à mes promoteurs T. Carletti et A. Sartenaer pour leurs nombreux retours et précieux conseils qui m'ont permis de mener ce travail jusqu'au bout.

Je tiens également à remercier mes parents ainsi que mon frère qui ont été là pour moi pendant toute cette aventure. Un grand merci aussi à Laurent qui m'a proposé de relire mon mémoire. Enfin, je tiens à remercier tout particulièrement Mathilde pour m'avoir donné la force et le courage nécessaire quand j'en avais besoin.

Résumé

Ce mémoire se fait en coopération avec le centre de recherche Cenaero et portera sur son logiciel d'optimisation Minamo. En particulier, ce dernier utilise un algorithme génétique, que nous appellerons GA, qui, en simulant une sélection Darwinienne, permet de résoudre différents problèmes d'optimisation avec et sans contraintes. C'est sur ce GA, et plus particulièrement sur la gestion de la taille de sa population que nous travaillerons dans ce mémoire.

Initialement, cette dernière reste constante et notre objectif sera de mettre en place une modification dynamique de sa valeur lors de l'exécution du GA. Pour cela, nous nous inspirerons d'une méthode nommée "*One Fifth Success Rule*" qui sera adaptée pour l'algorithme génétique de Minamo. Avec cette nouvelle implémentation, nous espérons améliorer la qualité des solutions proposées par le GA de Minamo tout en améliorant la vitesse à laquelle ces solutions sont obtenues.

Pour la suite, nous testerons les versions modifiées de l'algorithme génétique de Minamo sur un ensemble de cas-tests afin de comparer leur performance par rapport à la version par défaut de ce GA. De cette manière, nous serons capables de voir en quoi la gestion dynamique de la taille de la population de cet algorithme modifie les résultats obtenus. Lors de nos analyses, nous chercherons également à déterminer les raisons qui expliqueraient les différences observées au niveau de ces résultats.

Grâce au travail réalisé dans ce mémoire, nous avons pu mettre en évidence l'intérêt d'utiliser une taille de population adaptative au sein de l'algorithme génétique de Minamo.

Abstract

This master thesis is carried out with the research center CENAERO and will focus on its optimization software MINAMO. In particular, this one uses a genetic algorithm, called GA, which, by simulating a Darwinian selection, makes it possible to solve different optimization problems with and without constraints. It is on this GA, and more precisely on the management of the size of its population that we will work in this master thesis.

Initially, this size remains constant and our purpose will be to set up a dynamic modification of its value during the GA's execution. To do this, we will be inspired by a method called "*One Fifth Success Rule*", which will be adapted for MINAMO's genetic algorithm. With this new implementation, we hope to improve the quality of the proposed solutions from the GA of MINAMO while also improving the speed at which these solutions are obtained.

In the following, we will test the modified versions of MINAMO's genetic algorithm on a set of test cases to compare their performance to the default version of this GA. In this way, we will be able to see how the dynamic management of the population's size of this algorithm modifies the obtained results. In our analyses, we will also try to determine the reasons that would explain the observed differences in these results.

Thanks to the work done in this master thesis, we were able to highlight the interest of using an adaptive population size within MINAMO's genetic algorithm.

Table des matières

Introduction

Dans notre société actuelle, de nombreux problèmes font appel à de l'optimisation pour être résolus et de nombreux algorithmes ont été mis au point dans ce domaine pour fournir une réponse à ces problèmes. Cependant, pour pouvoir être utilisée, une partie de ces méthodes demande des informations, typiquement sur le gradient ou le hessien, concernant la fonction objectif liée au problème d'optimisation à résoudre. Or dans certaines situations, ces données ne sont pas accessibles car cette dernière est trop complexe. De plus, il arrive aussi que des problèmes d'optimisation nécessitent d'être rapidement résolus et dès lors, nous ne pouvons pas toujours nous permettre d'attendre la véritable solution.

C'est pour pallier ces problèmes que plusieurs types d'algorithmes de recherche ont été mis au point. Nous nous intéresserons en particulier à ceux nommés algorithmes heuristiques. En général, ces derniers ne demandent que l'information sur l'évaluation de la fonction objectif pour fonctionner et leur but n'est pas d'atteindre la meilleure solution mais de fournir un résultat suffisamment acceptable dans une limite de temps imposée. Une classe particulière dans ces méthodes heuristiques est celle des algorithmes génétiques. Ceux-ci utilisent le principe de la sélection naturelle pour obtenir une solution acceptable et c'est sur cette classe d'algorithmes que nous allons travailler, en collaboration avec le centre de recherche Cenaero.

Les algorithmes génétiques possèdent de nombreux paramètres et si certains les fixent une fois pour toutes, d'autres les adaptent au cours de leur exécution. C'est sur cette nuance que nous allons travailler dans ce mémoire. En effet, nous chercherons à observer l'impact de cette adaptation dynamique de paramètres au travers de la méthode *One Fifth Success Rule* appliquée à la taille de la population sur l'algorithme génétique présent dans le logiciel Minamo, développé par Cenaero.

Dans le Chapitre 1, nous présenterons le contexte dans lequel nous travaillerons en définissant le formalisme lié à la résolution de problèmes d'optimisation. Ensuite, nous développerons les concepts généraux derrière la construction des algorithmes génétiques avant de décrire le fonctionnement de celui présent dans Minamo. Une fois cela fait, nous nous pencherons sur la problématique de la gestion des paramètres liés aux algorithmes génétiques et plus particulièrement sur celle de la taille de leur population. Par la suite, le Chapitre 2 présentera le centre de recherche Cenaero avec lequel nous avons collaboré pour ce mémoire. De plus, nous présenterons également dans ce chapitre le logiciel Minamo sur lequel nous avons travaillé pour mettre en place les différents algorithmes utilisés dans ce mémoire.

Une fois l'environnement dans lequel nous travaillerons bien défini, le Chapitre 3 présentera la méthode *One Fifth Success Rule* que nous explorerons dans ce mémoire. En particulier, nous développerons dans ce chapitre son mode de fonctionnement ainsi que

les adaptations réalisées pour pouvoir l'utiliser dans l'algorithme génétique de Minamo. Rappelons que ce mémoire cherche à explorer l'impact de cette méthode sur l'algorithme génétique de Minamo. Pour ce faire, le Chapitre 4 nous permettra de présenter les différents problèmes d'optimisation, appelés cas-tests, que nous utiliserons pour analyser cet impact. Après cela, nous définirons, dans le Chapitre 5, différents outils qui nous permettront de comparer les versions modifiées de Minamo avec celle par défaut. Ainsi, nous serons en mesure de déterminer si la gestion de la taille de la population avec la *One Fifth Success Rule* permet d'améliorer l'algorithme génétique de Minamo, que ce soit au niveau de la qualité de sa résolution ou par rapport à sa vitesse de résolution.

Nous utiliserons ces outils de comparaison dans les Chapitres 6 et 7 qui reprendront les différents résultats obtenus à partir des versions modifiées de Minamo présentées dans le Chapitre 3. Ces derniers nous permettront de visualiser comment l'utilisation de la *One Fifth Success Rule* impacte, de manière globale, les performances de l'algorithme génétique de Minamo. De plus, nous chercherons également dans ces chapitres à expliquer les différences que nous observerons entre les différentes versions de Minamo que nous testerons.

Finalement, nous clôturerons ce mémoire par un bref récapitulatif du travail réalisé pour ensuite terminer par quelques conclusions complétées de perspectives que ce mémoire a soulevées au cours de sa réalisation.

Chapitre 1

Mise en contexte

Sur base des références [1], [2] et [3], ce premier chapitre mettra en place la terminologie ainsi que le formalisme liés aux méthodes d'optimisation utilisées pour résoudre un problème donné. Nous introduirons également les algorithmes génétiques en présentant leur mode de fonctionnement général ainsi que la problématique sur laquelle nous allons travailler.

1.1 Optimisation : modélisation et formalisme

Dans un problème général d'optimisation, la méthode employée pour le résoudre dépend de la nature du problème. Cela mène naturellement à une première étape de modélisation. Cette dernière permettra de déterminer les différents composants qui seront utilisés pour représenter ce problème d'un point de vue formel.

Tout d'abord, il est nécessaire de déterminer la nature de ce qui doit être optimisé. Dans notre cas, nous cherchons à déterminer les valeurs pour des variables réelles appartenant à un espace de dimension n qui pourrait être supérieure à un. Ces variables seront alors modélisées par un vecteur noté x dont le nombre de composantes sera égal à la dimension du problème. Ensuite, ce que nous cherchons à optimiser sera représenté par une fonction objectif définie par

$$f : \mathbb{R}^n \to \mathbb{R},$$

où f ne possède pas de caractéristique particulière. Notez que certains cas d'optimisation demanderont que cette fonction soit de classe $\mathcal{C}^1$ voire même $\mathcal{C}^2$ mais ce ne sera pas notre cas. Dans un cadre général, nous chercherons toujours à minimiser f même si certaines situations auraient tendance à demander une maximisation de cette fonction. Ce choix n'est pas contraignant car un problème d'optimisation nécessitant une maximisation de notre fonction objectif est équivalent à une minimisation d'une autre fonction objectif. Cela provient du fait que nous savons que pour toute fonction g définie sur un domaine D,

$$\sup_{x \in D} g(x) = -\inf_{x \in D} \left(-g(x)\right).$$

De nombreuses situations demandent également de respecter plusieurs contraintes sur la solution recherchée et cela est modélisé par des fonctions contraintes dites d'égalité et d'inégalité. En général, ces dernières sont considérées comme étant au minimum continues. Nous

pouvons dès lors écrire la formulation mathématique de notre problème comme

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ s.c.\ & g_i(x) \leq 0 \ \forall i \in I \\ & h_j(x) = 0 \ \forall j \in E \\ & x_{inf} \leq x \leq x_{sup}, \end{aligned} \tag{1.1}$$

où f représente la fonction objectif à minimiser en tenant compte des contraintes d'inégalité modélisées par les fonctions g_i indicées par l'ensemble I et d'égalité via les fonctions h_j ayant comme indices ceux appartenant à l'ensemble E. Les symboles x_{inf} et $x_{sup} \in \mathbb{R}^n$ correspondent aux frontières imposées sur l'ensemble des solutions envisageables. Si la dimension du problème est supérieure à un, chaque composante du vecteur x sera sujette à une contrainte similaire. Autrement dit, nous travaillerons sur des fonctions qui ont des hyper-rectangles pour domaine. Dans la suite de ce mémoire, lorsque nous préciserons qu'une solution proposée pour le problème (1.1) est admissible, cela signifiera que cette dernière sera considérée comme respectant les contraintes imposées par les fonctions g_i et h_j tout en appartenant au domaine défini par la dernière ligne de (1.1).

1.2 Deux types de solution : locale et globale

Dans ce mémoire, nous nous focaliserons sur un ensemble de fonctions dont le domaine de définition est réel. C'est par rapport à ce type de domaine particulier que nous allons différencier les notions de solution locale et globale d'un problème d'optimisation. Ces dernières seront respectivement notées x_{loc} et x_{glob}.

Une solution locale du problème (1.1) signifie que dans un voisinage de cette solution, la fonction objectif est bien minimisée par x_{loc}. Autrement dit,

$$\exists V \text{ voisinage de } x_{loc} \text{ tq } \forall x \in V \cap R,\ f(x_{loc}) \leq f(x),$$

où R l'espace de recherche de f défini par son domaine de définition et les contraintes qui sont imposées. Dans certains problèmes d'optimisation, ce type de solution suffit mais dans le cadre de ce mémoire, nous désirons travailler sur des algorithmes ayant pour but de déterminer une solution globale au problème (1.1) ou au moins de s'en approcher.

Le concept de solution globale est assez simple : une fois cette dernière atteinte, nous avons la certitude que n'importe quel autre point, dans le domaine de la fonction objectif et respectant les contraintes, fournira une évaluation supérieure à celle obtenue via x_{glob}. Autrement dit, en reprenant la définition de l'espace de recherche R donnée précédemment, nous pouvons écrire que

$$\forall x \in R, f(x_{glob}) \leq f(x).$$

La Figure 1.1 permet de visualiser la différence entre le caractère local ou global d'une solution. De manière évidente, lorsqu'une solution est globale, elle est également locale mais la réciproque n'est pas vraie.

Les différentes méthodes mises au point pour obtenir une solution, qu'elle soit locale ou globale, peuvent être divisées en deux parties. D'un côté nous avons les méthodes déterministes et de l'autre les stochastiques. Une des différences qui permet de séparer ces deux

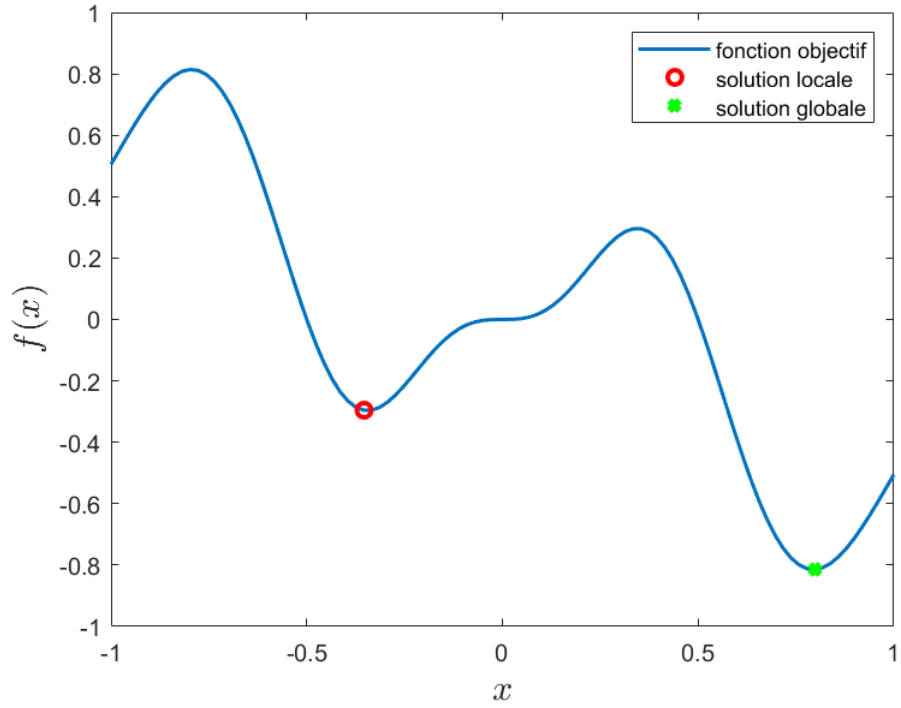


FIGURE 1.1: Illustration des deux types de solution pour le problème (1.1). Inspirée de [4].

types concerne les résultats obtenus lorsque ces méthodes sont utilisées plusieurs fois avec comme entrées les mêmes valeurs à chaque essai. Dans le cas des méthodes déterministes, le résultat obtenu après chaque exécution sera toujours identique tandis que pour les méthodes stochastiques, il sera différent à chaque itération. Cela est dû au fait que le deuxième type de méthodes utilise la notion de hasard dans son implémentation, ce qui amène forcément à des résultats différents malgré les mêmes entrées utilisées. C'est dans cette catégorie de méthodes que nous travaillerons.

1.3 Motivation des algorithmes génétiques

De nombreux algorithmes ont été mis au point afin d'obtenir la solution exacte, notée x^*, d'un problème d'optimisation. Nous pouvons en retrouver certains dans [1] tels que la méthode de la plus forte pente ou bien celle du gradient conjugué. Le point commun entre ces deux méthodes est qu'elles utilisent des informations sur la fonction objectif comme son gradient ou son hessien. Dans certaines applications concrètes demandant une optimisation, nous pouvons avoir suffisamment d'information concernant la fonction objectif pour pouvoir appliquer ce type de méthodes mais il peut arriver que ce ne soit pas le cas. La fonction objectif est alors considérée comme une "boite noire". Concrètement, cela signifie que nous ne connaissons que ce que nous injectons comme entrées pour la fonction et le résultat qui en découle. Dans ce cas, toute méthode faisant appel à des informations sur la fonction objectif comme son gradient n'est pas applicable. C'est dans ce domaine que nous nous plaçons et la solution utilisée sera celle des algorithmes évolutionnaires et plus particulièrement les algorithmes génétiques, notés GA.

Ces algorithmes font partie de la classe des méthodes heuristiques présentées par N.

Kokash dans [5]. La définition de l'adjectif heuristique donnée dans le site Larousse est la suivante, "Qui consiste ou qui tend à trouver" [6]. La caractéristique de ce type de méthodes est que la solution trouvée ne sera pas celle du problème mais une approximation. Ces méthodes auront plus tendance à parcourir l'espace de recherche afin de s'assurer du caractère global de la solution proposée. En plus des algorithmes génétiques, nous retrouvons dans les méthodes heuristiques celle du *Simulated Annealing* qui peut accepter des solutions qui sont moins optimales au cours de son exécution. L'algorithme du *Tabu Search* fait aussi partie des méthodes heuristiques et utilise de la mémoire pour éviter les optima locaux. Cette liste n'est évidemment pas exhaustive et d'autres algorithmes sont présentés dans [5] mais nous ne nous y attarderons pas.

Dans le cas de certaines méthodes heuristiques telles que les algorithmes génétiques, de nombreux paramètres doivent être définis. Ceux-ci sont cruciaux car ils déterminent l'exactitude de l'approximation proposée par rapport à la vraie solution ainsi que la vitesse de convergence vers cette approximation. La problématique liée au choix de ces paramètres sera abordée dans la Section 1.4.3. Pour information, certaines méthodes heuristiques utilisent elles-mêmes une heuristique sur leurs paramètres afin de déterminer la valeur ou combinaison optimale pour le problème à résoudre. Dans ce cas-ci, nous parlons de méthodes méta-heuristiques.

1.4 Les algorithmes génétiques

Cette section commencera par présenter le principe général de ce type d'algorithmes tout en introduisant les notions fondamentales utilisées dans l'implémentation de ces derniers. Ensuite, la méthode générale sera présentée et illustrée. Nous terminerons par poser la problématique qui sera approfondie dans ce mémoire. L'ensemble de ces concepts sont tirés des références [7] et [8]. Nous utiliserons également les diapositives utilisées dans [2] pour pouvoir rester cohérent avec le point de vue de Cenaero.

1.4.1 Idée générale et concepts

Comme son nom l'indique, un algorithme génétique va s'inspirer de la nature pour simuler une sélection naturelle Darwinienne entre des solutions potentielles pour obtenir, de manière itérative, la meilleure solution possible à la fin de son exécution. Avant de préciser la méthode générale de cette catégorie d'algorithmes, nous allons définir quelques notions qui seront utilisées dans la suite :

- **Un individu** représente une instance de solution potentielle pour le problème (1.1). Dans notre contexte réel, il s'agira donc d'un n-uplet dont les composantes seront réelles. Ces dernières représentent donc toutes les caractéristiques d'un individu et ce sont elles qui seront modifiées ou utilisées pour pouvoir générer un nouvel individu.
- **Une population** représente un ensemble d'individus. Elle est caractérisée par sa taille, notée μ_t où t fait référence à la t-ème génération de l'algorithme.
- **Une génération** est le terme utilisé pour représenter une itération d'un algorithme génétique. Au cours de cette dernière, l'objectif sera d'utiliser sa population actuelle pour en générer une nouvelle qui servira de base pour la prochaine génération.
- **Une évaluation** d'un individu est l'étape qui permettra d'attribuer un score à celui-ci dans le but de le comparer à ses congénères d'une même population. Cette évaluation

peut être faite directement via la fonction objectif et les contraintes mais peut également être réalisée par une fonction dite de *fitness*. Dans MINAMO, nous utilisons une fonction appelée *Global Objective* ou *GO* qui permet de considérer simultanément le résultat de la fonction objectif et le respect des contraintes. Nous considérons qu'une évaluation fournie par la *GO* est performante lorsque cette dernière sera petite. Ce critère sera utilisé par après pour comparer deux individus. Le but de la fonction *GO* sera de représenter au mieux la qualité d'un individu dans le contexte exprimé par (1.1). Dans un cadre général, nous emploierons le terme de *fitness* pour mentionner cette notion d'évaluation.

- **La sélection** est une étape qui consiste à comparer la *fitness* des individus d'une population donnée pour mettre en évidence les plus performants. Pour ce faire, nous comparerons deux individus au niveau de leur *fitness* tout en tenant compte de leur respect ou non des contraintes imposées par le problème. En particulier, la *fitness* utilisée par MINAMO prend déjà en compte le respect de ces contraintes. Dans notre contexte nous utiliserons le mode de sélection proposé dans [2] qui consiste en trois critères. Tout d'abord, le respect des contraintes est observé et si un des deux individus les respecte mais pas son vis-à-vis, alors ce sera toujours le premier qui sera sélectionné. Ensuite, lorsque les deux individus ne violent pas les contraintes imposées, nous sélectionnerons celui qui aura la meilleure valeur pour la fonction objectif. Dans le pire des cas où aucun des candidats ne respecte les contraintes du problème, celui qui en sera le moins éloigné sera celui sélectionné. Dans le chapitre suivant, nous expliquerons comment cette distance par rapport aux contraintes a été mise en place dans MINAMO.
- **Le croisement** entre deux individus, considérés comme parents, consiste en l'utilisation de leurs caractéristiques respectives pour créer un nouvel individu, nommé enfant. Dans notre contexte, il s'agira donc d'utiliser les composantes de chacun des deux parents pour en générer des nouvelles qui formeront alors les caractéristiques d'un nouvel individu. Typiquement, une technique de croisement consiste à travailler composante par composante et d'utiliser une combinaison linéaire pour obtenir celle correspondante de l'enfant. Notons que dans l'algorithme génétique de MINAMO, le taux de croisement est fixé à 100%. Cela signifie que tous les individus d'une population peuvent participer au croisement.
- **La mutation** d'un individu correspond au changement aléatoire des caractéristiques de ce dernier. Une méthode classique évoquée dans [7] et [9], qui s'adapte bien à notre contexte réel, consiste en l'ajout d'une perturbation aléatoire qui pourrait s'apparenter à un bruit. Typiquement, un ajout d'une valeur aléatoire suivant une loi normale pourrait être effectué sur chaque composante caractéristique d'un individu. Dans les algorithmes que nous utiliserons, cette opération pourra se produire exclusivement sur les individus générés par l'étape de croisement. Comme énoncé dans [9], cette opération permet de jouer sur l'équilibre entre l'exploitation, qui consiste à tenter de résoudre le problème depuis l'endroit où nous sommes, et l'exploration, qui favorise la diversification de la zone de recherche. Autrement dit, plus le taux de mutation est élevé, plus l'algorithme aura tendance à explorer son domaine de recherche mais au risque de passer trop vite à côté de la solution optimale. D'un autre côté, un taux trop faible risquerait de bloquer la recherche de la solution dans une zone non intéressante, comme le voisinage d'un optimum local. Dans l'algorithme génétique de MINAMO, ce taux est fixé à 1% et l'opérateur utilisé est la mutation de Gauss.

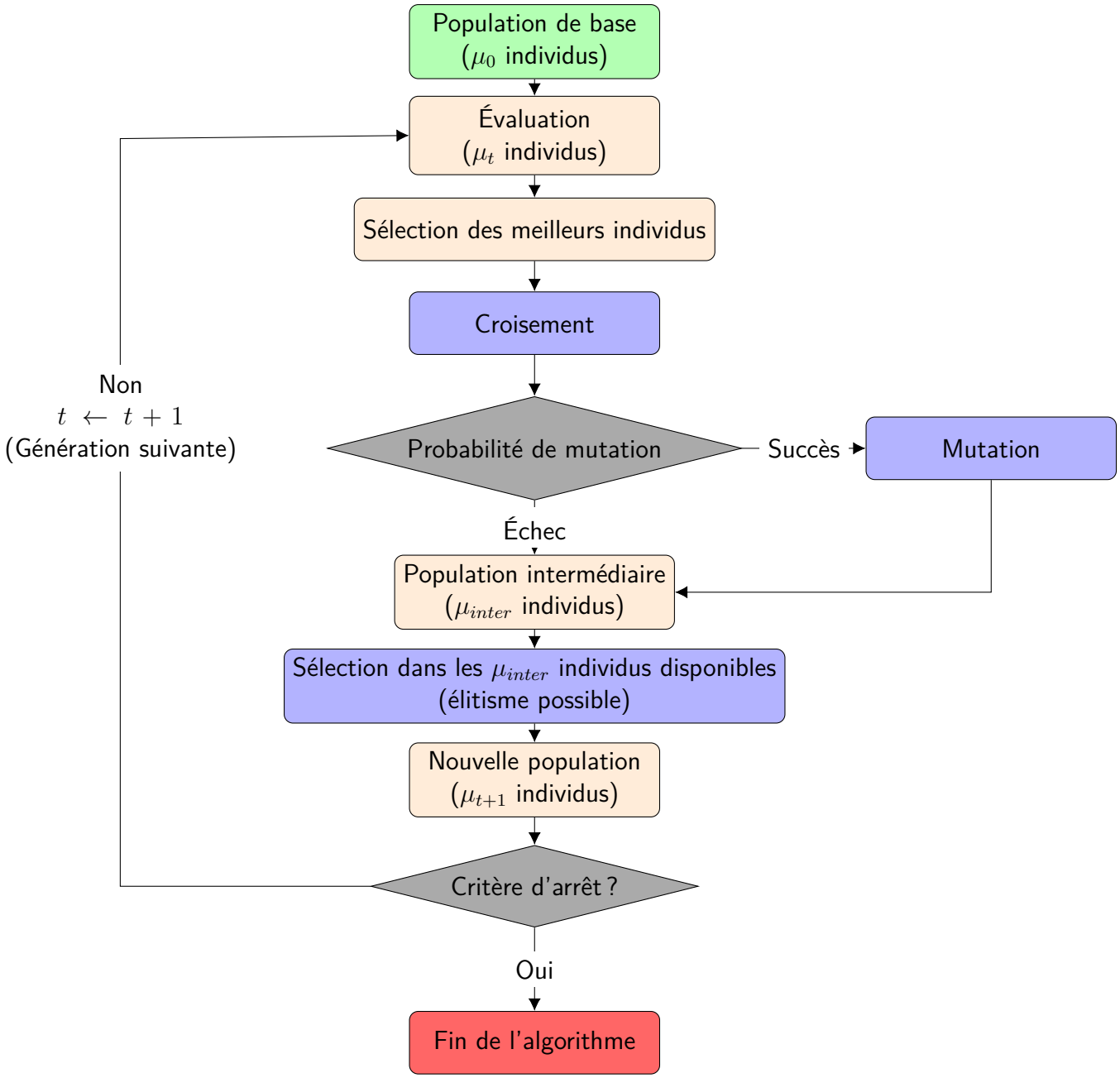


FIGURE 1.2: Représentation de la méthode général d'un algorithme génétique.

1.4.2 Mode de fonctionnement

Nous allons maintenant développer la méthode générale qui sera utilisée par un algorithme génétique pour résoudre un problème de type (1.1).

Tout d'abord, une population initiale de μ_0 individus est générée aléatoirement sur le domaine de définition de la fonction objectif f. Ensuite, chaque individu est évalué via la fonction de *fitness* pour déterminer sa performance. Une fois que cela est fait, l'étape de sélection permettra de mettre en évidence les meilleurs individus qui seront utilisés pour en créer de nouveaux. Chaque nouvel individu sera généré via le croisement entre deux parents faisant partie des individus sélectionnés à l'étape précédente. Lorsqu'un enfant a été généré, il est susceptible de subir une mutation qui permettra alors d'explorer davantage le domaine

de recherche. Cette étape fournira une population intermédiaire de μ_{inter} individus comprenant les parents et enfants. Afin de générer une nouvelle population de taille μ_{t+1} pour la génération suivante, une sélection devra être effectuée parmi l'ensemble des individus qui se basera sur leur évaluation. Il faudra donc évaluer les enfants avant de procéder à cette étape. Pour sélectionner un individu, l'algorithme se basera sur la qualité de son évaluation. Autrement dit, au plus la fonction de *fitness* fournira un résultat performant pour un individu, au plus ce dernier aura une chance d'être sélectionné pour faire partie de la nouvelle population. Comme cette étape se base sur une probabilité pour déterminer si un individu sera sélectionné, nous n'avons pas l'assurance que les meilleurs individus soient choisis pour faire partie de la nouvelle population. C'est pourquoi le principe d'élitisme peut être appliqué avant d'effectuer cette sélection. Il consiste à imposer le fait que les m meilleurs individus fassent partie de la nouvelle population. Dans l'algorithme de MINAMO, ce paramètre est fixé à deux. Dès lors, la sélection ne concernera plus que $\mu_{inter} - m$ individus. Dès qu'une nouvelle population a été générée, l'algorithme passe alors à la génération suivante et recommence ce processus, tant qu'un critère d'arrêt n'a pas été rencontré. Une illustration de ces étapes se trouve à la Figure 1.2. Notons que ce fonctionnement n'est pas commun à tous les algorithmes génétiques. Cependant, c'est celui qui correspond à l'algorithme génétique utilisé dans MINAMO et sur lequel nous travaillerons dans ce mémoire.

1.4.3 Problématique des paramètres

Comme décrit dans [7] et [8] à ce sujet, l'aléatoire possède une place importante dans ce genre de méthodes et certaines probabilités comme le taux de mutation ou bien de croisement peuvent être fixées comme paramètres lors de l'exécution de ce type d'algorithme. De nombreux autres paramètres doivent être également mis en place et cela varie en fonction du type d'algorithme génétique qui est implémenté. La véritable problématique ne réside pas en la détermination des paramètres mais plutôt autour du traitement qui leur sera accordé au cours de l'exécution d'un algorithme génétique.

L'article présenté dans [10] fournit un excellent point de vue sur cette problématique et peut être complété par A. Aleti dans la Section 2.4 de sa thèse [11]. Dans ces différentes sources, nous apprenons que le premier objectif dans ce domaine a été de déterminer un ensemble de paramètres fixes qui convenaient à un grand nombre de problèmes résolus par un algorithme génétique. Autrement dit, les paramètres étaient fixés une fois pour toute et n'étaient jamais modifiés au cours de l'exécution. Au terme de cette première vague de recherche menée par De Jong dans [12] et Grefenstette dans [13], des solutions ont été exposées mais des problèmes plus récents ont montré le manque d'efficacité de cette utilisation statique de paramètres. Il a alors semblé logique de travailler sur des méthodes permettant une adaptation dynamique de ces derniers.

Nous utiliserons le classement et la terminologie fournie dans [10] et [11] pour différencier les types de méthodes utilisés. Par souci de compréhension, nous ne considérerons que le cas où un seul paramètre doit être adapté. Premièrement, nous avons les méthodes déterministes. Celles-ci se caractérisent par le type de règles utilisées pour modifier le paramètre concerné. Ces dernières ne prennent pas en compte l'état de la recherche de solution pour décider de la modification ou non du paramètre. Autrement dit, lors de l'implémentation d'une telle méthode, c'est un élément déterministe qui sera pris en compte pour décider si le changement du paramètre devra être effectué ou non. Typiquement, nous pouvons considérer

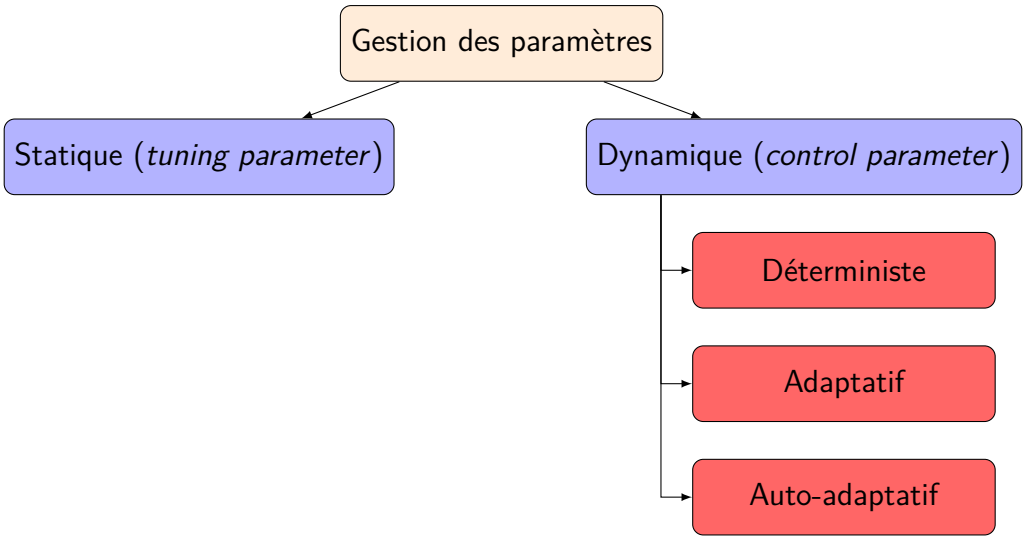


FIGURE 1.3: Illustration des différentes méthodes pour la gestion des paramètres dans un algorithme génétique. Inspirée de [10] et [11].

le nombre de générations passées dans l'algorithme et imposer un changement du paramètre qui sera effectué toutes les k générations. Ensuite viennent les méthodes adaptatives. Dans ce cas-ci, le critère mis en place pour adapter notre paramètre utilisera de l'information provenant de la recherche de solution toujours en cours. Par exemple, un critère de performance pourrait être fixé et à chaque génération de l'algorithme, si ce critère est respecté par la population, le paramètre en question sera modifié. Pour finir, nous avons la catégorie dite auto-adaptative. Ici, le paramètre à adapter sera incorporé dans le domaine de recherche du problème d'optimisation et il sera donc optimisé en même temps que la recherche de la meilleure solution au problème initial. L'ensemble de ces catégories de gestion des paramètres dans un algorithme génétique est représenté par la Figure 1.3.

Faisons maintenant un focus sur le paramètre gérant la taille de la population dans un algorithme génétique. Nous nous baserons sur [11] pour traiter plus en détails les difficultés liées à ce cas de figure.

1.4.4 Cas du réglage de la taille de population

Si nous devions simplifier le type de choix à faire pour le réglage d'une population fixée, nous aurions soit celui d'en prendre une de grande taille, soit en choisir une ayant un plus petit nombre d'individus. Dans le premier cas, une population possédant une grande taille sera plus susceptible d'amener à une exploration de l'espace de recherche. En effet, plus nous avons d'individus dans la nouvelle population, plus nous avons de chance que l'un d'entre eux subisse une mutation le menant vers une autre zone de l'espace de recherche. Avec le même raisonnement, le second cas rendra la recherche de solution plus locale car les chances qu'une mutation se produise seraient moins élevées avec peu d'individus.

La difficulté principale rencontrée lors du traitement de ce paramètre réside dans le fait que ce dernier est fort dépendant de la nature du problème d'optimisation à résoudre. En effet, si celui-ci est simple ou ne contient qu'un seul optimum (problème unimodal), la population nécessaire à l'utilisation d'un GA ne devra pas être trop importante dans le sens où

il ne sera pas nécessaire d'explorer beaucoup l'espace de recherche. Dans l'autre cas, il serait alors préférable d'avoir une population suffisamment importante pour pouvoir explorer l'ensemble des possibilités de l'espace de recherche et ne pas rester coincé dans un voisinage d'une solution locale. Malheureusement, il n'est pas toujours possible de connaitre autant d'informations sur la nature du problème à résoudre. De ce fait, fixer une fois pour toutes le paramètre gérant la taille de la population avant la mise en route d'un algorithme génétique risquerait de rendre ce dernier moins efficace. Il semble donc préférable de se tourner vers une gestion dynamique de ce paramètre.

Dans [14], Y. Derlet présente différentes méthodes utilisant cette gestion de paramètre pour la taille de la population. En particulier, la méthode dite *Saw-Tooth* consiste à décroitre de manière linéaire la taille de la population jusqu'à ce que cette dernière atteigne une valeur minimale. Une fois ce seuil atteint, la taille de la population est augmentée à sa valeur de départ et sera ensuite diminuée de la même manière lors des itérations suivantes. C'est donc une méthode qui fonctionne sur un principe de périodes. Dans son analyse, Y. Derlet [14] met en évidence l'efficacité de cette méthode par rapport à l'algorithme génétique présent dans Minamo. Les expérimentations faites dans [14] nous montrent que le temps d'exécution a été diminué de moitié grâce à la méthode *Saw-Tooth* tout en conservant des résultats proches, voire meilleurs de ceux obtenus avec l'algorithme génétique qui considère une taille de population fixe. Dans cette approche, nous retrouvons le besoin d'avoir une population importante en début d'exécution afin de favoriser l'exploration de l'espace de recherche et plus les générations passent, plus cette méthode se focalise sur l'exploitation en réduisant le nombre d'individus par population.

Grâce au travail de A. Aleti [11] et de Y. Derlet [14], nous voyons bien l'importance du rôle que joue la taille de la population dans un algorithme génétique. Une gestion intelligente de cette dernière permettra donc d'améliorer l'exactitude de la solution obtenue ainsi que la vitesse à laquelle nous l'obtenons. Il est important de noter que certaines méthodes présentées dans [14] ajoutent de nouveaux paramètres à gérer. En effet, pour la *Saw-Tooth*, le nombre de périodes optimal a été déterminé suite à des essais de valeurs différentes. Le danger serait alors de déplacer le problème du réglage de la taille de la population à celui des paramètres introduits par une nouvelle méthode. Dans cette optique, le travail réalisé dans ce mémoire peut être considéré comme un complément à celui fait dans [14]. Nous utiliserons une règle appelée *One Fifth Success Rule* pour gérer la taille de la population. Cette dernière sera présentée en détails dans la suite de ce mémoire mais nous pouvons déjà dire qu'elle utilisera l'état de la recherche en cours pour déterminer comment la population devra évoluer en cours d'exécution. De ce fait, là où la méthode *Saw-Tooth* peut être considérée comme une gestion dynamique déterministe, la *One Fifth Success Rule* fait plutôt partie des méthodes adaptatives, c.f. Figure 1.3. Comme l'objectif final de ce mémoire consiste à utiliser cette méthode dans le logiciel Minamo de Cenaero, nous allons décrire ce centre de recherche ainsi que la particularité de l'algorithme présent dans ce logiciel avant d'introduire la *One Fifth Success Rule*.

Chapitre 2

Présentation de Cenaero et Minamo

Ce chapitre va dans un premier temps introduire le centre de recherche Cenaero avec qui nous collaborons pour ce mémoire sur base des informations recueillies dans [15] et [16]. Une telle présentation a également été faite dans les mémoires [14] et [17]. Dans un second temps, nous expliciterons les concepts qui régissent le logiciel sur lequel nous adapterons la méthode *One Fifth Success Rule*. En plus du travail effectué par J. Blanchard et Y. Derlet dans leurs mémoires, nous nous baserons sur [2] et [18] pour cette partie.

2.1 Le centre de recherche Cenaero

Fondé en 2002 et situé à Gosselies, ce centre de recherche a pour objectif de développer et fournir des outils de simulations performants pour soutenir des entreprises dans la création de produits compétitifs. Il a d'abord été spécialisé dans l'aéronautique avant de se diversifier dans les domaines des transports terrestres, énergétiques, de l'environnement et de la santé.

Pour répondre à ses objectifs en matière de simulation, Cenaero a développé trois logiciels permettant de répondre à des demandes bien précises : Argo, Morfeo et Minamo. Le logiciel Argo a été mis au point pour les contextes nécessitant une simulation en mécanique des fluides. Morfeo est un outil permettant la modélisation de procédés de fabrication comme l'usinage ou le soudage. Quant à Minamo, il a été mis au point pour résoudre des problèmes de conception faisant appel à de l'optimisation. C'est avec ce dernier logiciel que nous allons travailler dans l'objectif de lui appliquer la *One Fifth Success Rule*.

La force de ce centre de recherche se situe dans sa capacité de calcul. En effet, Cenaero possède dans son infrastructure un supercalculateur nommé Zenobe. Ce dernier est constitué de deux parties dont la mise en place a débuté en 2011 pour se finir en 2015. Suite à cela, il est devenu le supercalculateur *Tiers-1* de Wallonie et possède environs 14000 cœurs. Sa capacité de calcul l'a positionné à la 300ème place en novembre 2014 parmi les 500 calculateurs mondiaux les plus puissants. Ce dernier sera utilisé pour ce mémoire afin de tester les différentes implémentations que nous mettrons en place dans le chapitre suivant. En particulier, les présents travaux ont bénéficié de moyens de calcul mis à disposition sur le supercalculateur Tier-1 de la Fédération Wallonie-Bruxelles, infrastructure financée par la région wallonne sous la convention n°1117545.

2.2 Le logiciel MINAMO

Ce logiciel utilise le principe d'algorithme génétique pour résoudre un problème d'optimisation tel que celui défini par (1.1). La motivation derrière ce choix réside dans le fait que la plupart des fonctions objectif à minimiser venant de l'industrie sont dites "black-box". Cela signifie que l'expression analytique de ces dernières n'est pas connue et donc, les seules informations disponibles sont celles concernant les arguments utilisés comme entrées et le résultat obtenu. Impossible alors de considérer des méthodes demandant des informations sur le gradient ou le hessien pour résoudre ces problèmes. Dès lors, le choix d'un algorithme génétique ne nécessitant que les évaluations de la fonction objectif apparait pertinent. Malheureusement, le choix d'un tel algorithme amène à un autre problème, celui du nombre élevé d'évaluations de la fonction objectif et des contraintes à respecter. En effet, dans le monde de l'entreprise, le coût temporel des évaluations de fonctions peut s'avérer important. C'est pourquoi MINAMO utilise une méthode basée sur des approximations de la fonction objectif et des contraintes appelées méta-modèles. Ce seront ces derniers que MINAMO cherchera à minimiser via un algorithme génétique. Cette méthode se nomme *optimisation assistée par méta-modèle* ou *Surrogate-Based Optimization* en Anglais et sera notée *SBO* dans la suite. Il ne faut pas perdre de vue que MINAMO est aussi capable d'effectuer une optimisation directement sur la fonction objectif sans passer par des méta-modèles mais dans le monde de l'entreprise, c'est généralement la méthode *SBO* qui est utilisée pour les raisons citées précédemment.

2.2.1 Fonctionnement de l'algorithme *SBO*

Comme dit plus haut, cette méthode utilise une approximation de la véritable fonction objectif f dont l'évaluation sera moins coûteuse, nous la noterons $\tilde{f}$. Dans un cadre général, les fonctions représentant les contraintes du problème seront également approximées par $\tilde{g}_i$ pour celles d'inégalité et par $\tilde{h}_j$ pour celles d'égalité. Pour ce faire, nous avons besoin de connaitre certains points de l'espace de recherche qui seront évalués par f ainsi que par les fonctions qui représentent les contraintes du problème décrites dans (1.1). Ces points formeront le *Design of Experiments* et serviront de base pour la construction des méta-modèles. En effet, une fois cette étape réalisée, l'information contenue dans le *Design of Experiments* sera incorporée dans une base de données qui sera utilisée et mise à jour dans la suite de l'algorithme pour construire des méta-modèles qui correspondront de plus en plus au problème de départ. Lorsque les premiers méta-modèles ont été construits, un algorithme génétique est appliqué au problème d'optimisation résultant afin de minimiser la fonction objectif approximée. Le résultat de cet algorithme génétique, noté $\tilde{x}$, sera ensuite évalué par f ainsi que par les fonctions g_i et h_j afin d'enrichir la base de données. Grâce à ces nouvelles informations, de nouveaux méta-modèles plus performants en termes d'approximation de f ainsi que des g_i et h_j seront construits à l'itération suivante et le processus sera répété tant qu'un critère d'arrêt n'aura pas été atteint. Dans le cas de MINAMO, il s'agira d'un nombre maximal d'itérations qui sera fixé à priori. La représentation schématique de cette méthode est disponible à la Figure 2.1.

Dans ce mémoire, nous chercherons à améliorer l'algorithme génétique qu'utilise le logiciel MINAMO au sein de sa boucle *SBO*. En d'autres termes, nous n'utiliserons pas l'algorithme *SBO* lors de notre travail car nous nous concentrerons sur l'amélioration de l'algo-

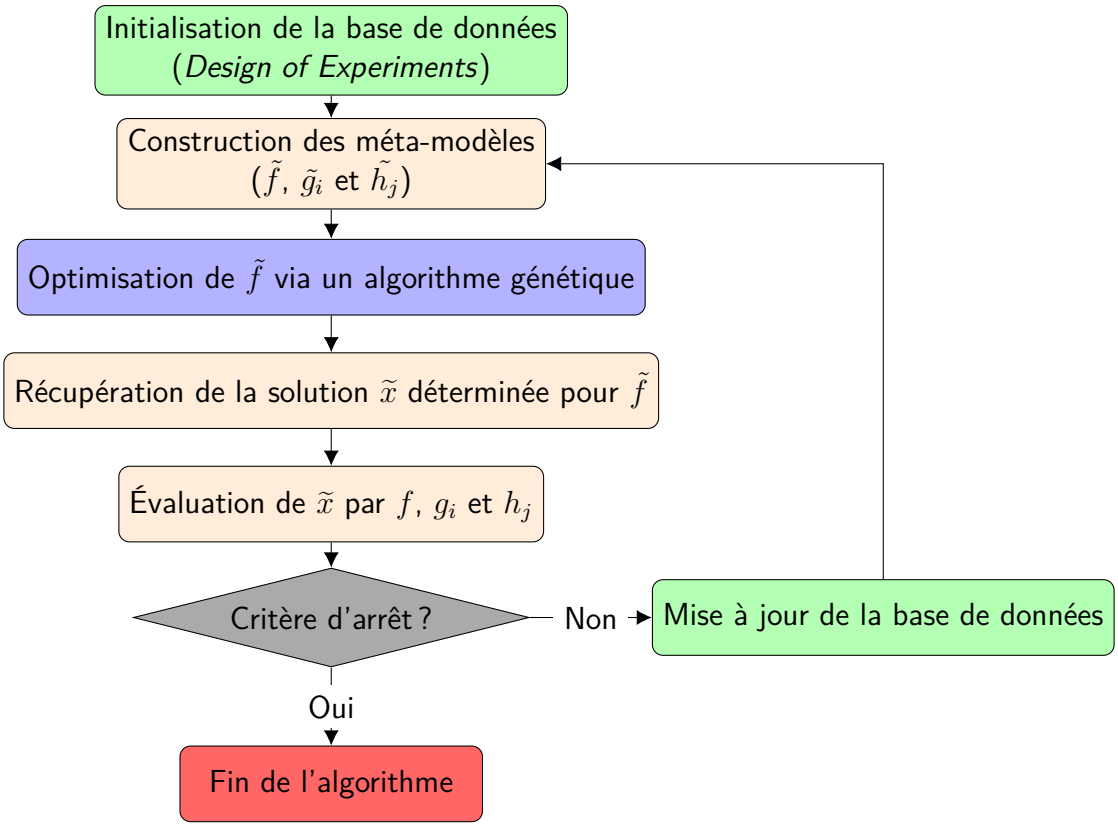


FIGURE 2.1: Schéma de l'algorithme *SBO* dans MINAMO, inspiré par [2], [14] et [17].

rithme génétique qui lui sert d'outil. Dès lors, l'ensemble des résultats qui seront présentés au fil de ce mémoire pourront être utilisés à l'avenir pour voir l'impact des changements effectués au niveau de l'algorithme *SBO* de MINAMO. Avant de présenter la méthode de la *One Fifth Success Rule* qui sera utilisée, nous terminerons ce chapitre par expliciter quelques particularités de l'algorithme génétique de MINAMO. En particulier, nous mentionnerons comment sa fonction de *fitness*, appelée *Global Objective*, a été pensée et nous traiterons également la gestion des contraintes faite par cet algorithme.

2.2.2 Particularités de l'algorithme génétique de MINAMO

Comme nous l'avons mentionné en début de Section 1.4.1, l'évaluation des individus dans cet algorithme se fait via une fonction particulière nommée *Global Objective* ou *GO*. Cette dernière fournit pour chaque individu une valeur plus grande ou égale à zéro. Le but de cette fonction est de prendre en compte, de manière simultanée, la valeur de la fonction objectif par rapport à l'optimum recherché et le respect des contraintes imposées par le problème. Pour ce faire, la construction de la fonction *GO* fournit différents paliers permettant de visualiser facilement la qualité de l'individu qui est évalué. En particulier, si cette valeur se trouve entre zéro et un, cela signifie que l'ensemble des contraintes est respecté par l'individu en question. Si la valeur est supérieure à un, cela signifie que l'individu n'est pas admissible. Il existe également des paliers supérieurs pour faire référence à des erreurs ou échecs lors du calcul de la fonction objectif f par la chaine de simulation mais nous ne les détaillerons pas ici. Ainsi, la construction de cette fonction *GO* implique que plus un individu présente une valeur élevée, plus ce dernier sera considéré comme mauvais par l'algorithme MINAMO.

Les informations disponibles dans [2] nous permettent de cerner comment les contraintes sont gérées dans MINAMO. En particulier, une valeur que nous noterons Σ_c sera calculée pour chaque individu x par

$$\Sigma_c(x) = \sum_{g_i(x)>0} W_i \left\| g_i(x) \right\| + \sum_{h_j \neq 0} W_j \left\| h_j(x) \right\|,$$

où les fonctions g_i et h_j correspondent aux contraintes du problème général d'optimisation (1.1). Autrement dit, la valeur de Σ_c est construite comme étant la somme de la norme des différentes fonctions contraintes qui n'auraient pas été respectées. Cette norme est définie, pour un individu x, par

$$\left\| c(x) \right\| = \left(c(x) - K \right)^2,$$

où $c(\cdot)$ est une fonction contrainte quelconque et où K est la valeur que cette fonction doit respecter. Si $c(\cdot)$ est une contrainte d'inégalité, K représentera la borne relative à cette dernière. En plus de sa norme, chaque contrainte sera associée à un poids particulier W_i ou W_j car il peut arriver que dans un problème d'optimisation, certaines contraintes s'avèrent être plus importantes que d'autres. La valeur de Σ_c sera égale à zéro lorsque l'ensemble des contraintes sera respecté et si ce n'est pas le cas, une valeur positive sera fournie. Cette dernière servira donc de mesure pour déterminer à quel point la violation des contraintes est intense ou non. Ainsi, si deux individus présentent une valeur de Σ_c supérieure à zéro, ce sera celui avec la plus grande valeur qui sera considéré comme le moins admissible.

Le chapitre suivant présentera la méthode que nous chercherons à explorer dans ce mémoire ainsi que son adaptation et implémentation dans l'algorithme génétique de MINAMO.

Chapitre 3

La *One Fifth Success Rule* (OFSR)

Nous allons commencer par nous baser sur un des plus simples algorithmes évolutionnaires pour saisir le fonctionnement de cette méthode. Ensuite, nous l'adapterons pour pouvoir l'appliquer efficacement à la gestion de la taille de la population dans l'algorithme génétique de Minamo. Notons que dans la suite de ce mémoire, nous utiliserons les termes *One Fifth Success Rule* et OFSR pour désigner cette méthode. L'ensemble des notions utilisées dans ce qui suit provient des articles [19] et [20].

3.1 L'algorithme $(1+1)$-ES

Commençons par fixer les notations qui seront utilisées dans la suite. Dans ce cas-ci, nous ne considérerons qu'un seul individu par génération et non une population. Nous noterons la dimension du problème par la lettre n et nous représenterons les estimations de la solution par $X_t \in \mathbb{R}^n$ avec t représentant la $t^{\text{ème}}$ génération, itération dans ce cas-ci, de l'algorithme. L'enfant produit à l'itération t sera écrit $\tilde{X}_t$ et est obtenu via l'équation

$$\tilde{X}_t = X_t + \sigma_0 \, \mathcal{N}(0, I_n). \tag{3.1}$$

Dans cette dernière, nous utilisons une distribution normale multivariée ayant le vecteur nul pour moyenne et une matrice de covariance égale à l'identité de dimension n. Nous voyons également que notre perturbation est pondérée par le terme σ_0 représentant la longueur du pas dans la direction déterminée par $\mathcal{N}(0, I_n)$ pour toute itération t. Ce coefficient scalaire est un paramètre à fixer lors de l'implémentation de cette méthode.

Le fonctionnement de cet algorithme (aussi appelé recherche aléatoire) est le suivant : nous commençons avec une solution de départ et une longueur de pas initialisées respectivement par X_0 et σ_0. Ensuite un nouvel individu est généré via (3.1) et sera comparé avec son parent via la fonction objectif. Cela permettra de déterminer lequel des deux est le plus performant et sera conservé pour l'itération suivante. Un individu sera plus performant que l'autre lorsque son évaluation par la fonction objectif sera inférieure à celle obtenue par l'autre individu. Cette opération sera ensuite répétée tant qu'un critère d'arrêt n'aura pas été rencontré, typiquement une limite sur le nombre d'itérations. Une illustration de cet algorithme est faite à la Figure 3.1.

Dans cette première version, nous remarquons que le paramètre représentant la longueur du pas reste constant tout au long de l'algorithme. La *One Fifth Success Rule* va imposer un

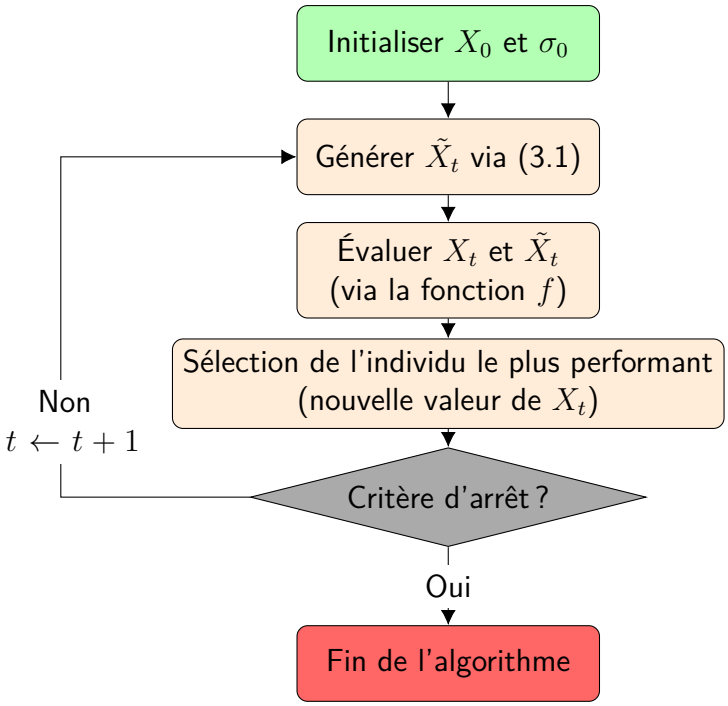


FIGURE 3.1: Schéma de l'algorithme $(1+1)$-ES sans adaptation de la longueur de pas.

changement sur ce paramètre en fonction du résultat de l'enfant par rapport à son parent. Ce changement est effectué à chaque itération t et est défini par

$$\begin{cases} \sigma_{t+1} = \alpha\sigma_t & \text{si } f(\tilde{X}_t) \leq f(X_t) \\ \sigma_{t+1} = \alpha^{-1/4}\sigma_t & \text{sinon,} \end{cases} \tag{3.2}$$

où α est un paramètre à valeur réelle et où σ_t représente la longueur du pas pour l'itération t. Dans [20], nous apprenons que les valeurs idéales pour ce paramètre se trouvent entre $2^{\frac{1}{n}}$ et 2 avec n la dimension du problème. Cela signifie que le terme α sera toujours strictement supérieur à un. En effet, nous pouvons démontrer le résultat suivant.

Résultat 3.1. *Pour tout entier naturel n supérieur ou égal à un, le terme $2^{\frac{1}{n}}$ est strictement plus grand que un.*

Preuve
Soit n, entier naturel quelconque fixé supérieur ou égal à un. Nous obtenons trivialement que

$$0 < \frac{1}{n}.$$

Or la fonction 2^x est strictement croissante. Par définition, nous avons donc que

$$\begin{aligned} &2^0 < 2^{\frac{1}{n}} \\ \Longleftrightarrow &1 < 2^{\frac{1}{n}}. \end{aligned}$$

Ce qui prouve bien le Résultat 3.1. □

Auger choisit dans son article [19] la valeur 1.5 pour α que nous conserverons dans la suite de ce mémoire. L'idée de cette définition (3.2) donnée dans [19] vient du fait que nous recherchons une solution globale au problème (1.1). Dès lors, si trop d'itérations successives

diminuent l'évaluation de la fonction objectif, cela signifierait que nous nous trouverions dans une zone menant à un optimum qui pourrait être local. Pour éviter ce cas de figure, la longueur du pas sera augmentée afin de favoriser une exploration de l'espace de recherche plutôt qu'une exploitation. D'un autre côté, si de nombreuses itérations successives ne permettent pas de diminuer la valeur de la fonction objectif, alors cela signifierait simplement que la longueur du pas est trop importante. Autrement dit, l'exploration de l'espace de recherche empêcherait de s'approcher d'un optimum. Dans ce cas, le pas verra sa longueur diminuée pour pouvoir se concentrer sur une zone plus précise de l'espace de recherche. Pour compléter cette première explication, l'article [21] indique que pour la plupart des fonctions non-linéaires, le taux d'amélioration de la solution, via la modification présentée dans (3.1), est une fonction monotone décroissante dépendant du paramètre σ. Cela implique que ce taux tendra vers zéro lorsque σ tendra vers l'infini. Suivant cette logique, il serait judicieux de contrôler ce paramètre afin de maitriser l'évolution de la solution dans l'algorithme $(1+1)$-ES. C'est-à-dire que la longueur du pas σ sera diminuée lorsque l'algorithme n'arrive pas à proposer une meilleure solution pour le problème considéré. De cette manière, nous augmentons alors le taux d'amélioration de la solution afin d'avoir plus de chance d'améliorer la recherche d'une meilleure solution. Le pseudo-code de la méthode $(1+1)$-ES utilisant la définition (3.2) est disponible dans l'Algorithme 1.

Regardons le lien entre cette méthode et la notion de "un cinquième". Pour ce faire, notons p_{ok} la probabilité de générer un enfant meilleur que son parent. Dès lors, comme réalisé par Auger, nous pouvons expliciter l'espérance concernant la valeur de la longueur du pas à la prochaine itération en connaissant celle de l'itération courante. Cette dernière peut se visualiser au travers de l'Algorithme 1 et est donnée dans [19] par

$$E(\sigma_{t+1}|\sigma_t) = \alpha^{p_{ok}} \left(\alpha^{-1/4}\right)^{1-p_{ok}} \sigma_t, \tag{3.3}$$

où σ_t représente la longueur du pas à l'itération courante et σ_{t+1} celle du pas à l'itération suivante.

En observant l'équation (3.3), nous pouvons en distinguer trois cas correspondant à la valeur de p_{ok}. Considérons d'abord le cas où la probabilité p_{ok} est égale à un cinquième.

Algorithme 1 : Pseudo-code pour $(1+1)$-ES avec la *One Fifth Success Rule*

Initialiser $X_t \leftarrow X_0$, $t \leftarrow 0$ et $\sigma_t \leftarrow \sigma_0$
tant que $t < nbrIterMax$ **faire**
  $\tilde{X}_t = X_t + \sigma_t \mathcal{N}(0, I)$ ▷ Création d'un nouvel individu
  si $f(\tilde{X}_t) \leq f(X_t)$ **alors**
    $X_t \leftarrow \tilde{X}_t$ ▷ Remplacement du parent par l'enfant
    $\sigma_t \leftarrow \alpha\sigma_t$ ▷ Augmentation du pas
  fin
  sinon
    $\sigma_t \leftarrow \alpha^{-1/4}\sigma_t$ ▷ Diminution du pas
  fin
  $t \leftarrow t + 1$
fin

Cela implique que

$$\begin{aligned} E(\sigma_{t+1}|\sigma_t) &= \alpha^{1/5}\left(\alpha^{-1/4}\right)^{1-1/5}\sigma_t \\ &= \alpha^{1/5}\alpha^{-4/20}\sigma_t \\ &= \alpha^{1/5-1/5}\sigma_t \\ &= \sigma_t. \end{aligned}$$

En d'autres termes, avoir une probabilité d'obtenir un enfant meilleur que son parent égale à un cinquième ne modifierait pas la longueur du pas pour l'itération suivante. Si maintenant la valeur de p_{ok} est différente de un cinquième, observons les options possibles pour le terme

$$\alpha^{p_{ok}}\left(\alpha^{-1/4}\right)^{1-p_{ok}}$$

issu de (3.3). Dans le cas où la valeur de p_{ok} serait strictement inférieure à un cinquième, nous aurions que

$$\alpha^{p_{ok}}\left(\alpha^{-1/4}\right)^{1-p_{ok}} \leq 1.$$

Dès lors, la longueur du pas sera réduite si la probabilité de générer un meilleur enfant est inférieure à un cinquième. En tenant un raisonnement similaire, nous pouvons voir que le pas sera augmenté si cette probabilité est supérieure à un cinquième. Le nom de la *One Fifth Success Rule* vient donc du fait que la valeur "un cinquième" représente le seuil permettant de savoir l'évolution du pas dans le cas de l'algorithme $(1+1)-ES$. L'ensemble de ces cas peut être résumé par

$$\begin{aligned} \text{si } p_{ok} < \frac{1}{5} &\Rightarrow \text{ Le pas est réduit,} \\ \text{si } p_{ok} = \frac{1}{5} &\Rightarrow \text{ Le pas n'est pas modifié,} \\ \text{si } p_{ok} > \frac{1}{5} &\Rightarrow \text{ Le pas est augmenté.} \end{aligned}$$

Des variantes de ces algorithmes existent et sont énoncées dans [19]. Ces dernières traitent du cas de figure où les évaluations du parent et de son enfant aboutissent au même résultat. Maintenant que la présentation du principe de la OFSR a été faite, nous allons en proposer une adaptation afin qu'elle puisse correspondre au réglage de la taille de la population dans un algorithme génétique. En particulier, nous travaillerons sur son implémentation dans l'algorithme génétique de Minamo.

3.2 La OFSR dans l'algorithme génétique de Minamo

Suite à nos recherches, nous n'avons pas trouvé de référence dans la littérature portant sur l'ajout de cette méthode dans un algorithme génétique. C'est pourquoi cette section va présenter la manière dont nous avons implémenté la méthode de la OFSR au sein de Minamo.

Avant toute chose, rappelons que pour un problème donné par (1.1), l'algorithme Minamo cherche à minimiser une fonction appelée *Global Objective* (*GO*). Cette dernière permet de considérer l'optimisation de la fonction objectif f et le respect de contraintes simultanément. Dès lors, Minamo possède déjà une structure implémentée qui met en

avant si une nouvelle génération d'individus qui vient d'être évaluée permet d'améliorer la recherche. Cette notion de performance est déterminée de la manière suivante, si un des nouveaux individus évalués possède une valeur de *GO* inférieure à celle du meilleur individu de la génération précédente, alors MINAMO considère que cette nouvelle génération est plus performante que la précédente et donc, que la recherche est améliorée. Cette notion de performance entre générations sera notre base pour pouvoir utiliser la OFSR via sa définition faite dans (3.2).

Pour pouvoir implémenter cette méthode dans MINAMO, nous avons mis en place des modifications juste après que la population à l'itération t ait été évaluée. De cette manière, nous connaissons déjà sa performance par rapport à celle de l'itération $t-1$. En s'inspirant du principe de la *One Fifth Success Rule*, nous avons défini que

$$\begin{cases} \mu_{t+1} = ceil(\alpha\mu_t) & \text{si l'itération } t \text{ améliore la recherche} \\ \mu_{t+1} = floor(\alpha^{-1/4}\mu_t) & \text{sinon,} \end{cases} \tag{3.4}$$

où μ_t représente la taille de la population à l'itération t et α est un paramètre réel fixé à 1.5 selon la référence [19]. Cette valeur ne sera pas modifiée par la suite. Les fonctions $ceil()$ et $floor()$ indiquent que leur argument sera arrondi, respectivement, à l'entier supérieur ou inférieur le plus proche. L'utilisation de ces fonctions est nécessaire car nous travaillons avec un nombre entier d'individus par population. Le choix de ces fonctions est motivé par la philosophie de la OFSR qui augmente la population si une amélioration est repérée et qui la diminue sinon. Dès lors, il est clair que les modifications que nous implémenterons, selon la définition (3.4), ne correspondront pas parfaitement à la *One Fifth Success Rule* présentée dans la Section 3.1. Toutefois, nous continuerons à employer ce terme pour faire le lien avec la méthode sur laquelle nous nous sommes inspirés.

Une fois la valeur pour μ_{t+1} définie, nous avons mis en place une vérification permettant de s'assurer que cette valeur ne devienne pas trop élevée ou trop petite. Le choix d'imposer ces bornes nous vient du fait que MINAMO fonctionne via un nombre fixé d'itérations qui lui sert de critère d'arrêt. Dès lors, la mise en place d'une borne supérieure pour la taille de la population permet de ne pas faire exploser le nombre d'évaluations par itération et la borne inférieure permet de toujours avoir une diversité suffisante parmi les individus. Dans la suite de ce travail, nous avons fixé la borne supérieure à

$$\mu_{sup} = 100n,$$

et la borne inférieure à

$$\mu_{inf} = 10n,$$

où n est la dimension du problème à optimiser. De plus, nous avons également imposé que la taille initiale de la population serait fixée à

$$\mu_0 = 30n,$$

afin que le point de départ des différentes méthodes soit similaire. En pratique, si la taille de la nouvelle population μ_{t+1} ne respecte pas ces différentes bornes, elle sera alors remise à sa valeur initiale. Ce choix a été fixé en se basant sur le travail réalisé par Y. Derlet dans [14]. Notons que lors de la première itération, nous maintiendrons la taille de la nouvelle population à la valeur μ_0 puisqu'il n'est pas encore possible de comparer la performance de

Algorithme 2 : Implémentation de la *One Fifth Success Rule* dans MINAMO

Initialiser $t \leftarrow 0$, $pop_t \leftarrow pop_0$ et $\mu_t \leftarrow \mu_0$
tant que $t < nbrIterMax$ **faire**
  Évaluer pop_t
  si $t \neq 0$ **alors**
    Détermination si meilleure performance via *GO*
    si pop_t *est meilleure que* pop_{t-1} **alors**
      $\mu_{t+1} \leftarrow \alpha\mu_t$
    fin
    sinon
      $\mu_{t+1} \leftarrow \alpha^{-1/4}\mu_t$
    fin
    si $\mu_{t+1} > \mu_{sup}$ *ou* $\mu_{t+1} < \mu_{inf}$ **alors**
      $\mu_{t+1} \leftarrow \mu_0$
    fin
  fin
  sinon
    $\mu_{t+1} = \mu_0$
  fin
  Génération de pop_{t+1} avec élitisme
  $t \leftarrow t + 1$
fin

la population courante avec la précédente.

Maintenant que la valeur pour μ_{t+1} est bien déterminée, nous avons utilisé une implémentation déjà présente dans MINAMO afin de générer le nombre d'enfants correspondant à μ_{t+1} pour obtenir la nouvelle population à évaluer. Notons que dans MINAMO, le principe d'élitisme est utilisé. Pour rappel, ce dernier consiste à garder dans la population $t + 1$ un nombre déterminé, ici égal à deux, des meilleurs individus de l'itération t et nous conserverons ce principe dans notre implémentation. Une fois que les enfants ont été générés suivant les opérations de croisement et de mutation définies dans la Section 1.4.1, ils seront évalués et la performance de la population $t+1$ par rapport à celle de l'itération t sera retenue avant de reproduire le même processus pour l'itération $t+2$. Cette première implémentation de la *One Fifth Success Rule* dans MINAMO peut être résumée dans l'Algorithme 2. Pour y faire référence par la suite, nous l'appellerons *GA_OneFifth*. Une fois cette première implémentation mise en place nous avons réfléchi à différentes variantes afin d'améliorer la stratégie de recherche de la OFSR.

3.3 Variantes de la OFSR pour MINAMO

Le principe général de ces différentes variantes consiste, dans un premier temps, à produire autant d'enfants, avec élitisme, qu'il y a d'individus dans la population t. Une fois cela fait, la taille de la nouvelle population sera calculée selon le même principe de performance présenté à la section précédente. Comme annoncé précédemment, si la nouvelle taille μ_{t+1} ne respecte pas les différentes bornes imposées, elle sera automatiquement remise à μ_0. Se-

lon que la valeur de μ_{t+1} est supérieure ou inférieure à μ_t, différentes actions seront effectuées.

Dans le cas où μ_{t+1} est supérieure à μ_t, plusieurs stratégies ont été envisagées pour ajouter des individus dans la nouvelle population afin que la taille prévue soit atteinte. Nous avons décidé que ces nouveaux individus seraient, soit des enfants supplémentaires générés à partir de la population t, soit des individus générés aléatoirement selon la méthode utilisée pour la génération de la population initiale. Une dernière variante consiste à générer la moitié des individus manquants en utilisant des enfants venant de la population t et à générer le reste des individus aléatoirement. Pour information, si le nombre d'individus manquants est impair, alors nous utiliserons plus d'enfants que d'individus générés aléatoirement pour remplir la population $t + 1$. Ces différentes versions **pour ajouter des individus** seront nommées :

- *Offspring* (*off*) pour la génération de nouveaux enfants uniquement ;
- *Infill* (*ifl*) pour la génération aléatoire d'individus uniquement ;
- *Both* (*2*) pour la combinaison des deux versions précédentes.

Pour la situation dans laquelle μ_{t+1} est plus petite que μ_t, nous avons mis en place deux stratégies principales pour supprimer des individus parmi les enfants générés initialement. Pour chacune d'elles, nous souhaitons utiliser le *GO* de ces enfants mais ils ne sont pas encore évalués à ce stade. Par conséquent, et uniquement dans ce cas de figure, les enfants générés au nombre de μ_t seront évalués à ce niveau et nous indiquerons par la suite qu'il ne sera pas nécessaire d'évaluer la population $t+1$ puisque ça aura déjà été fait. Une fois cette évaluation faite, une première stratégie pour la suppression d'individus consistera à effacer les enfants possédant une valeur de la fonction *GO* peu performante. La seconde stratégie reprend ce que Y. Derlet a présenté dans [14] pour la stratégie FiscIS_EA et consiste à calculer une probabilité de survie par individu afin de déterminer ceux qui seront conservés ou supprimés. La formule initiale de cette probabilité de survie, pour une population pop_t venant d'être évaluée, est donnée par

$$surv(ind) = \begin{cases} \frac{GO_{max} - GO(ind)}{GO_{max} - GO_{min}}, \text{ si } GO_{max} \neq GO_{min} \\ 1 \text{ sinon,} \end{cases} \tag{3.5}$$

où ind est un individu quelconque de la population pop_t avec GO_{min} et GO_{max} qui représentent respectivement la plus petite et la plus grande valeur du *global objective* parmi tous les individus de pop_t. Pour déterminer la survie des individus de pop_t, un nombre aléatoire r est tiré uniformément sur l'intervalle $[0, 1]$ afin d'être comparé à la probabilité de survie de chaque individu pour déterminer si ce dernier devait être conservé ou non. Pour ce faire, nous considérons qu'un individu ind survit lorsque

$$surv(ind) > r. \tag{3.6}$$

De ce fait, la définition de $surv(ind)$ donnée par (3.5) implique que plus $GO(ind)$ sera petit, plus la probabilité de survie de ind sera proche de un. Autrement dit, plus l'individu ind aura un meilleur *Global Objective*, plus sa probabilité de survie sera élevée. Tout comme cela a été mentionné dans [14], la formule pour calculer la probabilité de survie possède des variantes utilisant la moyenne ou bien la médiane de l'ensemble des valeurs de *GO* de la population impliquée que nous utiliserons comme différentes versions dans notre implémentation. Ces variantes de (3.5) sont définies par

$$surv(ind) = \begin{cases} \frac{1}{2} + \frac{1}{2}\frac{ref - GO(ind)}{ref - GO_{min}} \text{ si } GO(ind) \leq ref, \\ \frac{1}{2}\frac{GO_{max} - GO(ind)}{GO_{max} - ref} \text{ sinon,} \end{cases} \tag{3.7}$$

où *ref* vaut soit la médiane, soit la moyenne des valeurs de *GO* pour la population pop_t. De plus, il a également été discuté du choix entre fixer le nombre r pour l'ensemble de la population ou le déterminer pour chaque individu. Ces deux options ont également été considérées dans notre implémentation. Dès lors, les différentes possibilités pour **la suppression d'individus** lorsque μ_{t+1} est plus petit que μ_t seront appelées

- *GO* (*go*) lorsque nous supprimons les individus avec le moins bon *global objective*,
- *Classic* (*cls*) lorsque nous utilisons la probabilité de survie définie par (3.5),
- *Mean* (*mn*) lorsque nous employons (3.7) avec *ref* égal à la moyenne des *GO*,
- *Median* (*med*) lorsque nous employons (3.7) avec *ref* égal à la médiane des *GO*.

Notons également que pour chaque version utilisant la probabilité de survie, nous considérerons le cas où le nombre aléatoire r est tiré une seule fois pour toute la population et celui où il sera généré pour chaque individu. Ces deux possibilités seront marquées respectivement par les notations *fx* et *vr*.

Pour résumer ces différentes variantes, nous générons, dans un premier temps, un nombre d'enfants égal à μ_t. Ensuite, la taille de la nouvelle population μ_{t+1} est calculée en fonction de la performance de la population actuelle t par rapport à la précédente. Par la suite, l'ajout ou la suppression d'individus se fera selon les différentes stratégies énoncées précédemment et chaque combinaison de ces dernières correspondra à une variante de la *One Fifth Success Rule*. L'Algorithme 3 permet de visualiser comment nous avons implémenté ces différentes versions dans Minamo.

Dans la suite de ce mémoire, nous chercherons à déterminer, parmi ces variantes, quelles seront celles que nous retiendrons comme les meilleures afin de les confronter avec la version initiale de la OFSR et l'algorithme par défaut de Minamo. Pour ce faire, nous allons définir un ensemble de problèmes d'optimisation, que nous appellerons cas-tests, sur lesquels nous nous baserons pour effectuer nos comparaisons.

Algorithme 3 : Implémentation des variantes de la *One Fifth Success Rule* dans MINAMO

Initialiser $t \leftarrow 0$, $pop_t \leftarrow pop_0$, $evalReq \leftarrow true$ et $\mu_t \leftarrow \mu_0$
tant que $t < nbrIterMax$ **faire**
  si $evalReq$ **alors**
    Évaluer pop_t
  fin
  Détermination si meilleure performance
  Génération de μ_t enfants avec élitisme
  si $t \neq 0$ **alors**
    si pop_t *est meilleure que* pop_{t-1} **alors**
      $\mu_{t+1} \leftarrow \alpha\mu_t$
    fin
    sinon
      $\mu_{t+1} \leftarrow \alpha^{-1/4}\mu_t$
    fin
    si $\mu_{t+1} > \mu_{sup}$ *ou* $\mu_{t+1} < \mu_{inf}$ **alors**
      $\mu_{t+1} \leftarrow \mu_0$
    fin
    si $\mu_t < \mu_{t+1}$ **alors**
      Ajouter les $\mu_{t+1} - \mu_t$ individus manquants ▷ choix entre *ifl*, *off* ou *2*
    fin
    sinon
      Évaluation des μ_t enfants
      Suppressions de $\mu_t - \mu_{t+1}$ enfants ▷ choix entre *go*, *mn_fx*, *cls_fx*,
      ▷ *med_fx*, *mn_vr*, *cls_vr* ou *med_vr*
      $evalReq \leftarrow false$
    fin
  fin
  $t \leftarrow t + 1$
fin

Chapitre 4

Présentation des cas-tests utilisés

Dans le chapitre précédent, nous avons mis en évidence différentes versions de la *One Fifth Success Rule*. Pour pouvoir déterminer quelle sera la version que nous retiendrons pour la suite de ce mémoire, nous observerons leurs résultats sur différents cas-tests. Ces derniers seront divisés en deux catégories, ceux qui ne présentent pas de contrainte et ceux qui en imposent. Pour tous ces cas-tests, nous présenterons leur définition ainsi que la valeur de leur solution globale. De plus, nous donnerons quelques caractéristiques sur leur fonction objectif respective.

4.1 Cas-tests sans contrainte

Tous les cas-tests présentés dans cette section sont issus de la littérature et peuvent être trouvés dans [14], [17], [22], [23] et [24].

- **Ackley_10**
  Ce cas-test possède comme fonction objectif la fonction Ackley de dimension $n = 10$. Sa définition analytique est donnée par

$$f(x) = f(x_1, \ldots, x_n) = -20\ exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \mathrm{e}$$

  où e vaut $exp(1)$ et n est la dimension de son domaine avec $x \in [-3, 2]^n$. Son optimum global se trouve en $x^* = (0, \ldots, 0)$ et vaut $f(x^*) = 0$. De plus, notons que cette fonction est fortement multimodale et non convexe.

- **Rastrigin_10**
  Pour ce cas-test, nous considérerons la fonction Rastrigin à optimiser. Cette dernière est définie de manière générale par

$$f(x) = f(x_1, \ldots, x_n) = 10\ n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right]$$

  où n est la dimension du domaine de cette fonction tel que $x \in [-5, 5]^n$. En particulier, nous considérerons le cas où $n = 10$. Tout comme la fonction précédente, l'optimum global se trouve en $x^* = (0, \ldots, 0)$ et vaut $f(x^*) = 0$. De plus la fonction Rastrigin est également fortement multimodale.

- **Rosenbrock_10**
  La fonction objectif à minimiser pour ce cas-test sera la fonction Rosenbrock de dimension dix. La définition de cette dernière est donnée par

$$f(x) = f(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

  où n est le nombre de composantes de x avec $x \in [-2, 2]^n$. Dans notre cas où n vaut dix, cette fonction est multimodale et son optimum global, situé en $x^* = (1, \ldots, 1)$ vaut à nouveau $f(x^*) = 0$. De plus, elle est non convexe comme la fonction Ackley.

- **Schwefel_10**
  Pour ce cas-test, nous travaillerons avec la fonction Schwefel dont la définition nous est fournie par

$$f(x) = f(x_1, \ldots, x_n) = 418.9828n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$$

  où n est la dimension du domaine de cette fonction avec $x \in [-500, 500]^n$. Dans ce cas-test, la valeur de n est fixée à dix. Tout comme les fonctions précédentes, celle-ci est multimodale avec un optimum global en $x^* = (420.9687, \ldots, 420.9687)$ où $f(x^*)$ vaut zéro. De plus, cette dernière est également non convexe.

Comme vous avez dû le remarquer, nous avons choisi de fixer la dimension des cas-tests de cette section à dix. Ce choix arbitraire est motivé par l'envie de travailler sur des cas-tests sans contraintes présentant un peu de difficultés mais sans aller dans l'extrême. Dès lors, nous avons choisi comme compromis de fixer leur dimension à dix.

4.2 Cas-tests avec contraintes

Ces cas-tests avec contraintes ont été sélectionnés parmi ceux mis à disposition par Cenaero. Les informations à leur sujet peuvent être trouvées dans [14], [17], [23] et [24].

- **G2_plog_10**
  Pour ce cas-test, la dimension n du problème à optimiser vaut dix et deux contraintes doivent être respectées. Tout d'abord, la fonction objectif $f(x)$ est définie par

$$f(x) = f(x_1, \ldots, x_n) = -\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

  où le domaine de cette fonction est tel que $x \in]0, 10]^n$. Les contraintes auxquelles ce problème d'optimisation est soumis sont données par

$$g_1(x) = \text{plog}\left(0.75 - \prod_{i=1}^{n} x_i\right) \leq 0$$

$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \leq 0,$$

avec la fonction plog(·) définie par

$$\text{plog}(\tilde{x}) = \begin{cases} \log(1+\tilde{x}), & \text{si } \tilde{x} \geq 0 \\ -\log(1-\tilde{x}), & \text{sinon} \end{cases}$$

où log représente la fonction logarithme en base dix. À notre connaissance, la meilleure solution connue pour ce cas-test vaut $f(x_{best}) = -0.74063$.

- **G7**
  Ce cas-test présente une fonction objectif à dix variables dont la définition est donnée par

$$\begin{aligned} f(x) = f(x_1, \ldots, x_{10}) = {} & x_1^2 + x_2^2 + x_1 x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\ & + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \end{aligned}$$

  où chaque variable x_i doit être dans l'intervalle $[-10, 10]$. De plus, huit contraintes doivent être respectées pour que la solution proposée soit réalisable. Ces dernières sont

$$\begin{aligned} g_1(x) &= -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 && \leq 0 \\ g_2(x) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 && \leq 0 \\ g_3(x) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 && \leq 0 \\ g_4(x) &= 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 && \leq 0 \\ g_5(x) &= 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 && \leq 0 \\ g_6(x) &= x_1^2 + 2(x_2 - 2)^2 - 2x_1 x_2 + 14x_5 - 6x_6 && \leq 0 \\ g_7(x) &= 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 && \leq 0 \\ g_8(x) &= -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} && \leq 0. \end{aligned}$$

  Tout comme pour le cas-test G2_plog_10, nous ne disposons que de la meilleure solution connue comme repère pour l'optimum global. Ce dernier vaut $f(x_{best}) = 24.30620$.

- **G10**
  Ce dernier cas-test est de dimension huit et va chercher à minimiser une fonction objectif qui n'utilise que trois variables. Cette dernière est donnée par

$$f(x) = f(x_i, \ldots, x_3) = x_1 + x_2 + x_3.$$

  Dès lors, ce sont les contraintes imposées qui vont utiliser l'ensemble des variables de ce cas-test. En particulier, ces dernières doivent être définies de sorte que

$$\begin{aligned} x_1 &\in [100, 10000] \\ x_2 \text{ et } x_3 &\in [1000, 10000] \\ x_i &\in [10, 1000]\,, \ \forall i = 4, \ldots, 8. \end{aligned}$$

Au niveau des contraintes, elles sont données par

$$\begin{aligned}
g_1(x) &= -1 + 0.0025(x_4 + x_6) && \leq 0 \\
g_2(x) &= -1 + 0.0025(x_5 + x_7 - x_4) && \leq 0 \\
g_3(x) &= -1 + 0.01(x_8 - x_5) && \leq 0 \\
g_4(x) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 && \leq 0 \\
g_5(x) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 && \leq 0 \\
g_6(x) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 && \leq 0.
\end{aligned}$$

À nouveau, l'optimum global que nous utiliserons comme référence sera le meilleure obtenu empiriquement. En particulier, la valeur de ce dernier sera donnée par $f(x_{best}) = 7049.24802$.

Cet ensemble de sept cas-tests sera celui que nous utiliserons dans la suite de ce mémoire pour pouvoir comparer les différentes versions de la *One Fifth Success Rule* présentées dans la Section 3.3. Une fois cela fait, nous confronterons celle que nous retiendrons à la version par défaut de Minamo ainsi qu'à celle initiale de la OFSR décrite dans la Section 3.2. C'est dans ce but que le chapitre suivant présentera les différents critères et outils que nous utiliserons pour effectuer cette comparaison.

Chapitre 5

Performance globale et outils d'analyse

Dans ce chapitre, nous présenterons la méthode utilisée dans la suite de ce mémoire pour pouvoir comparer la performance de différents algorithmes d'optimisation sur un ensemble donné de cas-tests. En particulier, cet outil nous servira à déterminer, parmi toutes les versions de la *One Fifth Success Rule* implémentées dans MINAMO, celle que nous considérerons comme la plus performante. Pour compléter cette analyse globale, nous fournirons également différentes informations quantitatives et graphiques qui permettront de comparer ces versions de manière plus détaillée pour chaque cas-test considéré.

Avant toute chose, rappelons que nos différents cas-tests cherchent à résoudre un problème d'optimisation défini par

$$\begin{aligned} &\min_{x \in Dom f} f(x) \\ \text{s.c } &g_i(x) \leq 0 \ \forall i \in I \\ &h_i(x) = 0 \ \forall i \in E \end{aligned} \tag{5.1}$$

où f est la fonction objectif, g_i sont les contraintes d'inégalité indexées dans I et h_i celles d'égalité dont les indices sont dans E. Pour chaque cas-test, nous avons défini une valeur comme étant celle de la fonction objectif en son optimum global qui sera notée $f(x^*)$ et appelée l'objectif. Afin de pouvoir analyser les résultats fournis par nos différentes versions, nous avons fixé une tolérance, propre à chaque cas-test, notée ε_f, telle qu'une version quelconque sera considérée comme ayant résolu son cas-test défini par (5.1) si elle arrive à fournir une solution $\tilde{x}$ telle que

$$\begin{aligned} &err_f(\tilde{x}) \leq \varepsilon_f \\ \text{et } &\Sigma_c(\tilde{x}) = 0, \end{aligned} \tag{5.2}$$

où $\Sigma_c(\tilde{x})$ défini la mesure de la violation des contraintes par $\tilde{x}$. Comme expliqué dans la Section 2.2.2, si cette mesure est nulle, cela signifie que $\tilde{x}$ respecte toutes les contraintes imposées. Pour la fonction $err_f(\tilde{x})$, cette dernière représente l'erreur absolue de la fonction objectif en $\tilde{x}$ avec la valeur de l'objectif $f(x^*)$. Elle sera définie par

$$err_f(\tilde{x}) = |f(\tilde{x}) - f(x^*)|, \tag{5.3}$$

pour toute solution $\tilde{x}$ proposée par un algorithme d'optimisation quelconque. De plus, afin de prendre en compte l'échelle de l'objectif à atteindre pour f, notre tolérance ε_f sera adaptée à cette dernière pour chaque cas-test. Notons que dans notre contexte où la fonction

objectif est bien définie analytiquement, nous chercherons toujours à respecter exactement les contraintes. Cependant, il peut arriver que dans un contexte industriel, une certaine tolérance sur la violation des contraintes soit mise en place mais cela ne nous concernera pas dans le cadre de ce mémoire.

Dès lors, si nous souhaitons comparer les valeurs de la fonction objectif venant de deux solutions $\tilde{x}_1$ et $\tilde{x}_2$ pour un même cas-test, celle que nous considérerons comme la meilleure sera celle possédant l'erreur absolue définie par (5.3) la plus petite. De plus, comme MINAMO cherche en premier lieu à produire une solution admissible, cela impliquera que pour deux solutions proposées $\tilde{x}_1$ et $\tilde{x}_2$, nous regarderons d'abord au respect des contraintes avant d'observer la valeur de $err_f(\cdot)$ pour ces solutions. Pour rappel, ce respect des contraintes est représenté par un poids noté $\Sigma_c(\tilde{x})$ qui vaut zéro lorsque toutes les contraintes sont respectées par $\tilde{x}$. Précisons que plus la valeur pour Σ_c est élevée, moins nous considérerons que les contraintes sont respectées. Par conséquent, et sans perdre de généralité, si la solution $\tilde{x}_1$ est celle présentant la plus grande valeur pour Σ_c, alors ce sera automatiquement la solution $\tilde{x}_2$ qui sera considérée comme la meilleure, même si la valeur de $err_f(\tilde{x}_1)$ est plus petite que celle de $err_f(\tilde{x}_2)$.

Dans le cadre de ce mémoire, nous analyserons la performance d'un algorithme selon trois points,

- la capacité ;
- l'efficacité ;
- la convergence.

Un algorithme sera considéré comme **capable** s'il parvient à résoudre, au sens de (5.2), le problème d'optimisation qui lui sera imposé. En ce qui concerne le point portant sur l'**efficacité**, nous considérerons qu'un algorithme v_1 sera plus efficace qu'un autre, noté v_2, si celui-ci parvient à résoudre le cas-test considéré en demandant moins d'évaluations des fonctions f, g_i et h_j définies par (5.1). En effet, nous avons considéré que ces dernières représenteraient le coût général d'un algorithme d'optimisation. De plus, si v_2 ne parvient pas à résoudre le cas-test concerné, alors v_1 sera directement considéré comme le plus efficace. Par rapport à la notion de **convergence**, nous regarderons simplement quelle sera la version qui arrive à se rapprocher le plus de son objectif via une solution admissible. Cette notion de proximité avec l'objectif sera représentée par l'erreur relative des valeurs de la fonction objectif avec ce dernier. Nous la noterons $rel_f(\tilde{x})$ et sa définition est donnée par

$$rel_f(\tilde{x}) = \begin{cases} \frac{|f(x^*)-f(\tilde{x})|}{|f(x^*)|} \text{ si } f(x^*) \neq 0, \\ |f(x^*) - f(\tilde{x})| \text{ sinon.} \end{cases} \tag{5.4}$$

Cette définition est utilisée pour ne plus tenir compte de l'échelle de l'objectif $f(x^*)$ recherché. Dès lors, au plus cette erreur sera petite et les contraintes respectées pour un algorithme, au plus nous considérerons que sa convergence est bonne.

Rappelons que nous travaillons avec des algorithmes stochastiques et par conséquent, deux exécutions d'une même version ne garantissent pas de fournir le même résultat. Cela est dû aux facteurs aléatoires que nous retrouvons dans la conception de ces algorithmes. Pour pouvoir analyser les résultats des différentes versions de la OFSR et également ceux de MINAMO, nous les avons donc exécutés chacun cent fois afin d'analyser la moyenne ainsi que la médiane des résultats obtenus. Dans la suite de ce mémoire, nous utiliserons le terme

run pour parler d'une exécution faite pour un algorithme quelconque. Passons maintenant à l'outil que nous utiliserons pour comparer la performance globale de différents algorithmes d'optimisation.

5.1 Profils de performance

Dans la littérature, cette question de performance globale a été notamment explorée par Dolan et Moré dans [25]. Ils y ont défini une méthode permettant de comparer, selon un certain budget, la capacité de différents algorithmes à résoudre un ensemble de problèmes d'optimisation, que nous appellerons cas-tests. Dans la suite, nous nommerons C l'ensemble de cas-tests sur lequel nous souhaitons comparer un ensemble d'algorithmes d'optimisation, appelé A. Pour rappel, le budget que nous avons choisi de considérer représente le coût en termes d'évaluation des fonctions f, g_i et h_j définies dans (5.1) alors que Dolan et Moré utilisent le temps de calcul comme budget. Comme le principe général de cette méthode de comparaison consiste à observer la proportion de cas-tests résolus pour chaque algorithme en fonction du budget accordé, le choix de ce dernier n'influence donc pas la suite de cette méthode. Dans notre contexte, nous considérerons qu'un cas-test sera résolu si la solution fournie respecte (5.2).

Dans [25], Dolan et Moré présentent différentes notions nécessaires pour construire cette méthode des profils de performance. Tout d'abord, nous avons besoin de connaitre le budget qui a été demandé par chaque algorithme de A pour résoudre l'ensemble des cas-tests considérés. Dans ce but, en reprenant ce que Dolan et Moré ont présenté, nous définissons pour tout cas-test $c \in C$ et pour tout algorithme $a \in A$,

$t_{c,a}$ = nombre médian d'évaluations nécessaires pour que l'algorithme a résolve le cas-test c.

Cette définition est motivée par le fait que chaque algorithme considéré sera exécuté cent fois. Dès lors, nous observerons pour chaque run le nombre d'évaluations nécessaires à la résolution de c et nous conserverons la médiane de nos résultats comme mesure du budget utilisé. En particulier, si un run de a n'arrive pas à résoudre c, alors $t_{c,a}$ sera fixé à l'infini. Par conséquent, plus la valeur de $t_{a,c}$ sera petite, plus nous considérerons que a est performant pour résoudre c. Le choix de la médiane est motivé par le fait que nous ne voulons pas que notre mesure soit influencée par des valeurs extrêmes. De ce fait, a ne sera pas pénalisé si certains runs isolés n'arrivent pas à résoudre c.

Suite à cette première définition, nous pouvons définir le ratio de performance pour a selon le cas-test c qui sera donné par

$$r_{c,a} = \frac{t_{c,a}}{\min\limits_{\tilde{a} \in A} t_{c,\tilde{a}}}. \tag{5.5}$$

Autrement dit, ce ratio consiste à comparer le budget demandé par l'algorithme a pour résoudre c par rapport à celui du meilleur algorithme de A. De cette manière, $r_{c,a}$ aura des valeurs qui se limiteront à l'intervalle $[1, +\infty[$ et plus sa valeur sera proche de un, plus il sera considéré comme meilleur. En particulier, si $r_{c,a} = 1$, cela signifie que l'algorithme a est celui qui demande le plus petit budget pour résoudre le cas-test c.

Avec cette notion de ratio, Dolan et Moré ont construit la définition du profil de performance pour un algorithme a qui est donnée par

$$\rho_a(\tau) = \frac{1}{|C|}|K_{\tau,a}|, \tag{5.6}$$

où

$$K_{\tau,a} = \{\tilde{c} \in C : r_{\tilde{c},a} \leq \tau\},$$

avec τ un nombre réel compris entre 1 et l'infini. Cela signifie que la valeur de τ représente une limite sur la proportion du budget que nous autorisons à l'algorithme a pour résoudre les cas-tests de C. Plus sa valeur est élevée, plus le budget accordé à a sera grand. Dès lors, le profil de performance $\rho_a(\tau)$ nous donne la proportion de cas-tests résolus par a avec cette limite de budget. Par conséquent, il sera positif pour un algorithme a d'avoir une valeur pour son profil de performance proche de un et ce, pour tout τ dans $[1, +\infty]$. En effet, cela signifie que l'algorithme a arrive à résoudre un grand nombre de cas-tests avec la contrainte sur le budget imposée par τ.

En particulier, lorsque τ est égal à un, les différents profils de performances nous permettent de déterminer les algorithmes de A les plus efficaces pour l'ensemble des cas-tests C. En effet, la valeur donnée par leur profil de performance nous indiquera la proportion de cas-tests résolus pour un budget minimal. Augmenter la valeur de τ nous permettra de visualiser cette proportion pour chaque algorithme lorsque nous leurs accordons de plus en plus de budget. En pratique, nous calculerons pour chaque algorithme de A leur profil de performance avec un τ allant de un jusqu'à une valeur maximale τ_{max} car accorder un budget infini n'est pas pertinent dans notre contexte.

Grâce à toutes ses notions issues de [25], nous avons pu obtenir un graphique, visible à la Figure 5.1, qui représente le profil de performance pour un ensemble d'algorithmes donnés par rapport aux cas-tests présentés dans le Chapitre 4. Sur l'axe des abscisses, nous pouvons y observer l'évolution de la valeur de τ pour laquelle chaque ordonnée correspondante nous indique la valeur $\rho_a(\tau)$ selon les différents algorithmes considérés. Comme nous l'avons expliqué précédemment, les courbes possédant l'ordonnée la plus élevée lorsque τ vaut un sont celles qui ont globalement été les plus efficaces pour résoudre les différents cas-tests. En particulier, nous pouvons voir que les algorithmes ifl_cls_fx et 2_cls_fx sont les moins performants. Plus précisément, le fait que leur ordonnée, pour $\tau = 1$, vaut zéro indique qu'aucun d'entre eux n'est l'algorithme qui demande le plus petit budget pour résoudre un cas-test. En revanche, nous pouvons observer que l'algorithme off_cls_vr est le plus efficace parmi ceux comparés puisque la valeur de son profil de performance est la plus élevée quand $\tau = 1$. Autrement dit, cet algorithme est celui qui résout le plus de cas-tests via un budget minimal. Pour rappel, le budget que nous considérerons dans ce mémoire sera représenté par le nombre médian d'évaluations nécessaires pour résoudre un cas-test.

La Figure 5.1 nous permet également de voir comment la performance des différents algorithmes évolue lorsque le budget accordé augmente. Par exemple, nous pouvons observer que même si 2_cls_fx était peu performant pour un petit budget autorisé, cet algorithme arrive à résoudre de plus en plus de cas-tests lorsque la limite sur le budget augmente. En particulier, l'algorithme ifl_cls_vr qui présentait une performance plus élevée pour un petit budget se fait surpasser lorsque la valeur de τ augmente. Par exemple, pour $\tau = 2$, nous pouvons voir que la valeur du profil de performance pour 2_cls_fx vaut plus du double

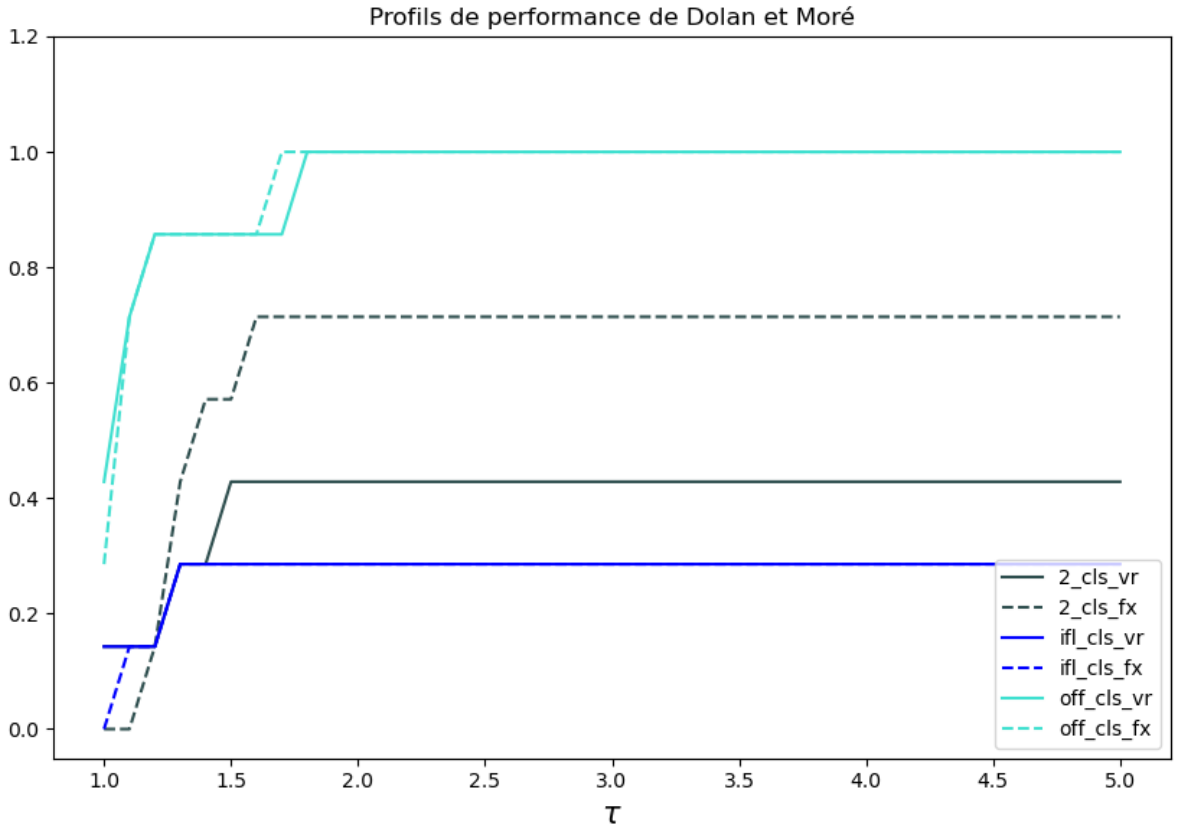


FIGURE 5.1: Graphique des profils de performance obtenus pour différentes variantes de la OFSR sur l'ensemble de cas-tests défini dans le Chapitre 4.

de celle pour ifl_cls_vr. Cela montre que l'algorithme 2_cls_fx est plus robuste que celui nommé ifl_cls_vr puisque lorsque nous augmentons le budget autorisé, le premier arrive à résoudre une plus grande proportion de cas-tests par rapport au second. En ce qui concerne les algorithmes les plus robustes, nous pouvons observer à la Figure 5.1 deux algorithmes dont le profil de performance arrive à atteindre la valeur un. Autrement dit, les algorithmes off_cls_vr et off_cls_fx sont les seuls qui parviennent, quand le budget est augmenté, à résoudre l'ensemble des cas-tests considérés. Parmi eux, l'algorithme off_cls_fx est celui qui demande le moins de budget pour atteindre ce résultat. En effet, il est celui dont le profil de performance demande le plus petit τ par rapport à off_cls_vr pour atteindre sa valeur maximale. Cependant, pour un budget limité au minimum, nous pouvons observer que c'est l'algorithme off_cls_vr qui arrive à résoudre une plus grande proportion de cas-tests. C'est pour cette raison que nous déciderons de considérer ce dernier comme étant, globalement, l'algorithme le plus performant parmi tous ceux considérés puisqu'il fait partie des plus robustes tout en étant le plus efficace.

Afin de compléter le résultat graphique de profil de performance, nous avons généré un tableau contenant différentes informations quantitatives issues du calcul de ce dernier pour chaque algorithme. En particulier, chaque ligne de ce tableau concernera un des algorithmes considérés, noté a, et contiendra

- la valeur calculée de $\rho_a(1)$;

version (acronyme)	$\rho_a(1)$	$\rho_a(\tau_{max})$	best τ
GA_Both_Classic (2_cls_vr)	0.14	0.43	1.50
GA_Both_Classic_fx (2_cls_fx)	0.00	0.71	1.60
GA_Infill_Classic (ifl_cls_vr)	0.14	0.29	1.30
GA_Infill_Classic_fx (ifl_cls_fx)	0.00	0.29	1.30
GA_Offspring_Classic (off_cls_vr)	0.43	1.00	1.80
GA_Offspring_Classic_fx (off_cls_fx)	0.29	1.00	1.70

Tableau 5.1: Résultats quantitatifs des profils de performance correspondant au graphique présenté à la Figure 5.1.

- la valeur calculée de $\rho_a(\tau_{max})$;
- la plus petite valeur de τ pour laquelle $\rho_a(\tau_{max})$ est atteint (intitulée "best τ").

Dès lors, nous connaitrons avec précision les valeurs des différents profils de performance lorsque le budget autorisé est minimal et maximal. De plus, la valeur représentée dans la colonne "best τ" nous permettra de voir à partir de quelle proportion du budget minimal la meilleure valeur du profil de performance est atteinte pour l'algorithme a. Le Tableau 5.1 présente ces résultats associés au graphique de la Figure 5.1. Il permet de compléter l'interprétation réalisée précédemment en indiquant de manière explicite les valeurs clés des profils de performance pour chaque algorithme. Typiquement, la colonne $\rho_a(1)$ indiquera le pourcentage de cas-tests pour lesquels l'algorithme considéré a été celui demandant le plus petit budget. En ce qui concerne la colonne $\rho_a(\tau_{max})$, elle nous donnera la proportion de cas-tests résolus par algorithme ,lorsque le budget accordé est maximal. Dès lors ces deux première colonnes nous indiqueront, quel est l'algorithme qui a été, respectivement, le plus efficace et le plus robuste. La dernière colonne "best τ" nous permettra de voir à partir de quelle proportion du budget minimal chaque algorithme arrive à atteindre son plus grand taux de cas-tests résolus. Pour lier ces résultats avec l'interprétation de la Figure 5.1 faite au paragraphe précédent, nous pouvons voir que ce sont bien les algorithmes off_cls_vr et off_cls_fx qui arrivent à résoudre tous les cas-tests considérés. De plus, comme nous l'avons déjà mentionné, bien que l'algorithme off_cls_fx demande moins de budget pour atteindre la valeur maximale de son profil de performance, nous considérerons que ce sera le off_cls_vr le plus performant puisque ce dernier est le plus efficace des deux, comme nous l'indique la colonne $\rho_a(1)$.

Dans la suite de ce mémoire, cet outil sera utilisé pour comparer les variantes de la OFSR définies dans la Section 3.3 afin de définir celle que nous conserverons pour la confronter avec l'implémentation initiale de la OFSR et l'algorithme par défaut de Minamo. Cependant, comme l'information fournie par les profils de performances est globale, cela ne nous permet pas d'effectuer une analyse plus détaillée pour comprendre le comportement des différentes versions de la *One Fifth Success Rule*. C'est pourquoi la section suivante présentera une série d'outils graphiques et quantitatifs propre à chaque cas-test afin de comparer en détails la performance des versions de la OFSR.

5.2 Outils de comparaison par cas-test

Cette section a pour but de présenter différents résultats propres à chaque cas-test considéré. Ces derniers ont été mis en place suite à la collaboration avec Cenaero pour pouvoir

comparer la performance de plusieurs algorithmes sur un cas-test en particulier et non de manière globale. Dès lors, ces derniers ne seront pas tous utilisés dans la suite de ce mémoire mais pourront toujours servir lorsqu'une analyse plus détaillée sera souhaitée.

Avant toute chose nous allons préciser quelles sont les données issues des exécutions de chaque algorithme que nous avions à notre disposition pour pouvoir générer ces différents résultats. En particulier, ces derniers sont fournis sous forme de liste reprenant différentes informations par rapport à chaque itération effectuée par le run considéré. En particulier nous avions, par itération

- la meilleure valeur de la fonction objectif : $f(\tilde{x}_{best})$;
- le score de contrainte de la meilleure solution proposée : $\Sigma_c(\tilde{x}_{best})$;
- le nombre d'individus dans la population : μ_t ;
- le nombre d'évaluations effectuées à cette itération : ω_t.

Par la suite, nous avons synthétisé ces différentes données afin de fournir un outil permettant d'analyser des algorithmes d'optimisation par cas-tests suivant les critères de **capacité**, de **convergence** et d'**efficacité** définis au début de ce chapitre.

Pour un cas-test quelconque, nous utiliserons un premier tableau qui indiquera le pourcentage d'exécutions ayant réussi à résoudre ce cas-test comme défini dans (5.2). Dès lors, ce tableau nous permettra de visualiser, pour chaque cas-test, quelle version présente le plus haut taux de réussite parmi l'ensemble de ses exécutions. Cela nous fournira donc une information sur la performance en terme de **capacité** pour chaque algorithme selon le cas-test concerné. Typiquement, plus ce pourcentage sera petit, moins la version sera considérée comme capable de résoudre le cas-test impliqué et inversement. Un exemple de ce type de résultat est visible dans le Tableau 5.2. Typiquement, ce dernier nous indique que les algorithmes off_cls_vr et off_cls_fx sont les plus à même de résoudre le cas-test considéré puisque leur pourcentage de runs réussis est supérieur à 70%. À contrario, nous pouvons dire que les algorithmes ifl_cls_vr et ifl_cls_fx ne sont pas capables de résoudre ce cas-test puisque leur taux de réussite est nul voire presque nul.

En ce qui concerne la convergence des différentes versions, nous utiliserons un tableau regroupant des informations statistiques par rapport à l'erreur relative, définie dans (5.4), des solutions finales fournies par les différents runs. Pour rappel, une bonne convergence est caractérisée par une petite erreur relative finale. Cependant, utiliser uniquement cette valeur ne suffit pas car certains cas-tests présentent des contraintes à respecter. Dès lors, une version sera performante du point de vue de sa **convergence** si elle réduit son erreur relative finale tout en égalant à zéro son score de pénalités Σ_c. C'est pourquoi nous avons

GA_Both_Classic (2_cls_vr)	16.0%
GA_Both_Classic_fx (2_cls_fx)	56.0%
GA_Infill_Classic (ifl_cls_vr)	0.0%
GA_Infill_Classic_fx (ifl_cls_fx)	1.0%
GA_Offspring_Classic (off_cls_vr)	72.0%
GA_Offspring_Classic_fx (off_cls_fx)	81.0%

Tableau 5.2: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test G10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_Classic (2_cls_vr)	[1.88e-01,2.56e+00]	1.05e+00	8.47e-01	0.00e+00
GA_Both_Classic_fx (2_cls_fx)	[7.40e-02,1.77e+00]	4.22e-01	3.85e-01	0.00e+00
GA_Infill_Classic (ifl_cls_vr)	[6.78e-01,2.93e+00]	1.74e+00	1.73e+00	1.08e-04
GA_Infill_Classic_fx (ifl_cls_fx)	[3.50e-01,2.76e+00]	1.67e+00	1.74e+00	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[1.27e-01,1.54e+00]	3.82e-01	3.20e-01	0.00e+00
GA_Offspring_Classic_fx (off_cls_fx)	[7.41e-02,1.21e+00]	3.32e-01	2.31e-01	0.00e+00

Tableau 5.3: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test G10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

ajouté à ce tableau un résultat statistique concernant ce score pour pouvoir correctement tirer des conclusions sur la convergence des algorithmes considérés. Pour récapituler, ce tableau contiendra pour chaque ligne

- le nom de la version considérée avec son acronyme ;
- l'intervalle de l'erreur relative finale avec l'objectif présentant la valeur minimale et maximale ;
- la moyenne de cette erreur relative finale ;
- sa médiane ;
- la moyenne du score de pénalité final ;
- la médiane du score de pénalité final.

Ainsi, l'analyse de ces différents éléments nous permettra de comparer la convergence des versions concurrentes que nous considérerons.

Pour compléter ces résultats tirés de nos différentes exécutions, nous avons utilisé le test statistique de Kruskal-Wallis présenté dans [26] suivi du test de Dunn, décrit dans [27], pour comparer statistiquement les médianes des erreurs relatives obtenues via les runs de chaque version considérée. Le premier test nous permettra de déterminer si au moins deux médianes, parmi celles considérées, sont significativement différentes. Si cela est confirmé par ce test statistique, alors nous utiliserons le test de Dunn qui nous permettra d'obtenir un tableau à double entrée permettant de voir exactement quels couples de versions présentent une différence significative entre leur médiane respective. En particulier, chaque couple de version sera associé à une valeur et si cette dernière est inférieure au seuil fixé à 0.05, alors nous considérerons que le couple concerné présente des médianes différentes à un degré de certitude de 95%. La combinaison du résultat de ces tests avec le tableau présenté au paragraphe précédent nous permettra de mettre en évidence, si elle(s) existe(nt), la ou les version(s) présentant la médiane concernant l'erreur relative la plus significativement petite.

Ces résultats sont illustrés dans les Tableaux 5.3 et 5.4. Le premier contient les différentes informations statistiques présentées précédemment et le second représente le résultat du test

	2_cls_vr	2_cls_fx	ifl_cls_vr	ifl_cls_fx	off_cls_vr	off_cls_fx
2_cls_vr	1.00e+00	3.42e-09	6.07e-06	4.91e-05	1.63e-12	1.23e-18
2_cls_fx	3.42e-09	1.00e+00	5.71e-29	6.12e-27	1.00e+00	8.43e-02
ifl_cls_vr	6.07e-06	5.71e-29	1.00e+00	1.00e+00	1.16e-34	1.89e-44
ifl_cls_fx	4.91e-05	6.12e-27	1.00e+00	1.00e+00	1.95e-32	6.34e-42
off_cls_vr	1.63e-12	1.00e+00	1.16e-34	1.95e-32	1.00e+00	1.00e+00
off_cls_fx	1.23e-18	8.43e-02	1.89e-44	6.34e-42	1.00e+00	1.00e+00

Tableau 5.4: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives reprises dans le Tableau 5.3 pour le cas-test G10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

de Dunn par rapport aux médianes des erreurs relatives. L'existence de ce second tableau implique donc que le test de Kruskal-Wallis a indiqué qu'il y a au moins deux algorithmes qui présentent des médianes de leurs erreurs relatives finales significativement différentes. En particulier, le Tableau 5.3 indique que les algorithmes 2_cls_fx, off_cls_vr et off_cls_fx ont une médiane qui est significativement plus petite que les autres. De plus, si nous observons les différents couples comprenant ces trois algorithmes dans le Tableau 5.4, nous pouvons voir que la valeur correspondante est supérieure à 0.05. Cela signifie que ces trois algorithmes ont des médianes de leurs erreurs relatives qui sont significativement identiques. En outre, comme la valeur médiane de leur score de pénalité est égal à zéro, nous pouvons considérer que ce sont ces trois versions qui ont, de manière équivalente, le mieux convergé. Notons également, qu'hormis un algorithme, tous ceux présents dans le Tableau 5.3 ont une valeur médiane pour Σ_c qui est nulle. Dès lors, les différents taux d'exécutions fructueuses visibles au Tableau 5.2 s'expliqueraient par le fait que certains algorithmes n'ont pas réussi à minimiser suffisamment la valeur de leur fonction objectif. Pour terminer cette analyse, nous pouvons faire un parallèle entre les Tableaux 5.2 et 5.3. En effet, les algorithmes présentant, dans le premier, les meilleurs taux de réussites correspondent à ceux, dans le second, dont la médiane des erreurs relatives est significativement la plus petite.

Pour analyser la performance en terme d'**efficacité** de différents algorithmes, nous utiliserons un ensemble de deux graphiques dont la lecture devra se faire parallèlement. Pour le premier, qui sera placé en haut, l'axe des abscisses représentera la médiane du nombre total d'évaluations effectuées par les différentes versions considérées. L'axe des ordonnées représentera la médiane des valeurs de la fonction objectif obtenues via les différentes exécutions. Dès lors, chaque courbe du graphique représentera l'évolution de la fonction objectif médiane par rapport au nombre médian d'évaluations effectuées pour la version correspondante. De plus, nous indiquerons également la borne supérieure de l'erreur absolue dépendante de la tolérance ε_f définie par

$$b_{sup} = f(x^*) + \varepsilon_f. \tag{5.7}$$

La borne inférieure de l'erreur absolue ne sera pas indiquée car nous travaillons dans un cadre où chaque problème d'optimisation demandera une minimisation de la fonction objectif. De ce fait, aucune valeur proposée par les différents algorithmes considérés ne pourra être strictement inférieure à l'objectif recherché. Ainsi, ce premier graphique nous permettra de déterminer quels sont les algorithmes qui ont, en médiane, atteint leur objectif en demandant le moins de budget, représenté par le nombre d'évaluations. Pour cela, il suffira

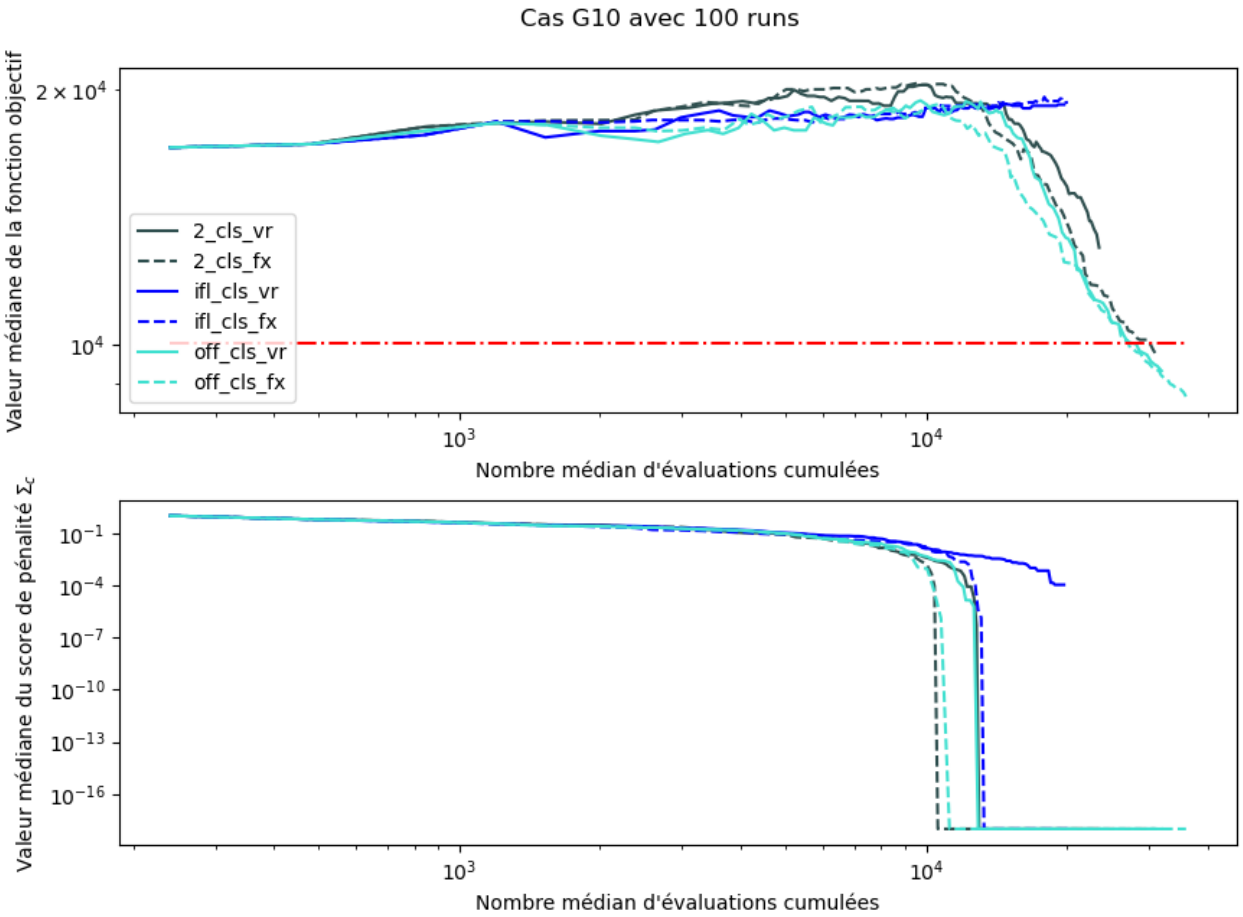


FIGURE 5.2: Graphiques présentant l'évolution de la valeur médiane de la fonction objectif (haut) et celle du score médian de pénalité (bas), toutes les deux par rapport au nombre médian d'évaluations effectuées. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7) pour le cas-test G10.

d'observer quelles sont les courbes qui atteignent b_{sup} en ordonnée pour une petite valeur de leur abscisse. Toutefois, nous ne pouvons pas visualiser à quelle vitesse les contraintes imposées par le cas-test considéré sont respectées, si elles le sont.

C'est pourquoi le graphique inférieur représentera en ordonnée la valeur médiane du score de pénalité Σ_c dont l'évolution sera également donnée en fonction du nombre médian d'évaluations. Ainsi, la lecture parallèle de ces deux graphiques nous permettra de déterminer quel est le budget médian demandé par chaque version pour résoudre l'ensemble des contraintes tout en diminuant suffisamment son erreur avec l'objectif. De plus, ces graphiques nous permettront également de visualiser quelles sont les versions qui n'ont pas réussi à résoudre le cas-test considéré. Typiquement, si une version présente une courbe de sa réponse médiane qui n'arrive pas à descendre sous b_{sup}, alors cette dernière n'aura pas réussi, selon la médiane, à résoudre son problème d'optimisation. Il en est de même si la courbe du score médian des pénalités n'arrive pas à atteindre zéro. Notons que ces deux graphiques seront également générés en considérant la moyenne de nos résultats plutôt que la médiane afin de visualiser la présence de valeurs extrêmes et leur impact si elles existent.

Dans la Figure 5.2, nous retrouvons un exemple de cette combinaison de graphiques.

Comme expliqué précédemment, cette dernière nous permet de donner des informations sur l'efficacité des différents algorithmes considérés. En particulier, si nous fixons la valeur des abscisses pour les deux graphiques visibles à la Figure 5.2, le graphique supérieur nous indiquera quelles sont les algorithmes dont la fonction objectif se rapproche suffisamment de l'objectif recherché. Typiquement, si nous fixons la valeur des abscisses à vingt-mille, nous pouvons voir qu'aucun algorithme ne s'est rapproché suffisamment de l'optimum global puisqu'aucune courbe n'a réussi à atteindre la borne de l'erreur supérieure. Cependant nous pouvons observer comment les algorithmes ont fait évoluer la valeur médiane le la fonction objectif jusqu'à cette abscisse. En effet, ceux dont la courbe correspondante est la plus basse sont ceux qui présentent la meilleure valeur médiane de la fonction objectif puisque nous cherchons toujours à effectuer une minimisation. En particulier, l'algorithme off_cls_fx est celui présentant la courbe la plus basse pour un nombre médian d'évaluations égal à vingt-mille. En ce qui concerne le graphique inférieur, il nous indique comment le score de pénalité a évolué avec le nombre d'évaluations. Notons que lorsque ce dernier était égal à zéro, nous avons modifié sa valeur en la fixant à 10^{-18} car l'échelle des ordonnées pour ces deux graphiques est logarithmique. Ainsi, pour une abscisse fixée à vingt-mille, il n'y a qu'un seul algorithme qui ne parvient pas à réduire à zéro son score de pénalité. Pour revenir au terme d'**efficacité** défini en début de ce chapitre, la Figure 5.2 nous montre que ce sont les algorithmes off_cls_vr et off_cls_fx qui atteignent les premiers la borne supérieure b_{sup}. De plus, en observant en parallèle le graphique concernant le score de pénalité médian pour l'abscisse correspondante, nous pouvons voir que ces deux algorithmes respectent toutes les contraintes du cas-test considéré. Notons qu'en plus de la notion d'efficacité, cette combinaison de graphiques permet de voir également la valeur médiane du nombre final d'évaluations effectuées par chaque algorithme. Pour cela, il suffit de regarder quelle est la valeur en abscisse que les différentes courbes atteignent.

Comme dernier outil d'analyse, nous utiliserons les résultats issus de nos différents runs pour générer un couple de graphiques permettant de mettre en parallèle l'évolution de la fonction objectif pour chaque algorithme avec celle de la taille de la population. Pour rappel, la version par défaut de l'algorithme génétique de Minamo génère un nombre constant d'individus à chacune de ses itérations. Avec la OFSR, nous faisons en sorte que ce nombre varie en fonction du résultat de la recherche. C'est pourquoi nous trouvons intéressant de visualiser, dans un premier graphique, l'évolution de la valeur médiane de la fonction objectif, sur les ordonnées, par rapport au nombre d'itérations des algorithmes considérés. Cette évolution sera mise en parallèle avec celle du nombre d'individus dans la population par rapport aux itérations de l'algorithme concerné que nous représenterons dans un second graphique. De cette manière, il nous sera possible de visualiser comment le nombre médian d'individus dans la population évolue et quel est l'impact de cette évolution au niveau de la valeur médiane de la fonction objectif pour chaque algorithme. Comme pour le résultat graphique précédent, nous avons également généré ces deux graphiques en utilisant la moyenne plutôt que la médiane.

Nous avons illustré ce dernier résultat à la Figure 5.3. En particulier, nous pouvons observer deux phases dans l'évolution de la population. Une première pendant laquelle cette dernière diminue pour tous les algorithmes et une seconde où tous augmentent progressivement leur population sauf deux. En utilisant également l'information reprise dans la Figure 5.2, nous pourrions associer la première phase avec la recherche de solution admissible et la seconde avec l'optimisation de la valeur de la fonction objectif. De plus, les graphiques de la

Figure 5.3 pourraient suggérer que pour le cas-test considéré, une augmentation progressive de la population permettrait d'améliorer la minimisation de la fonction objectif correspondante.

Pour résumer ce chapitre, nous avons mis en place un outil qui nous permettra de comparer la performance global des différentes versions de la OFSR entre elles et également avec la version par défaut de MINAMO. De plus, nous avons également mis à disposition une série d'outils permettant d'analyser plus en détails la performance de plusieurs algorithmes d'optimisation selon leur **capacité**, leur **convergence** et leur **efficacité**. Même si ces derniers ne seront pas tous exploités dans la suite de ce mémoire, nous trouvions intéressants de les présenter avec un exemple d'interprétation pour chacun.

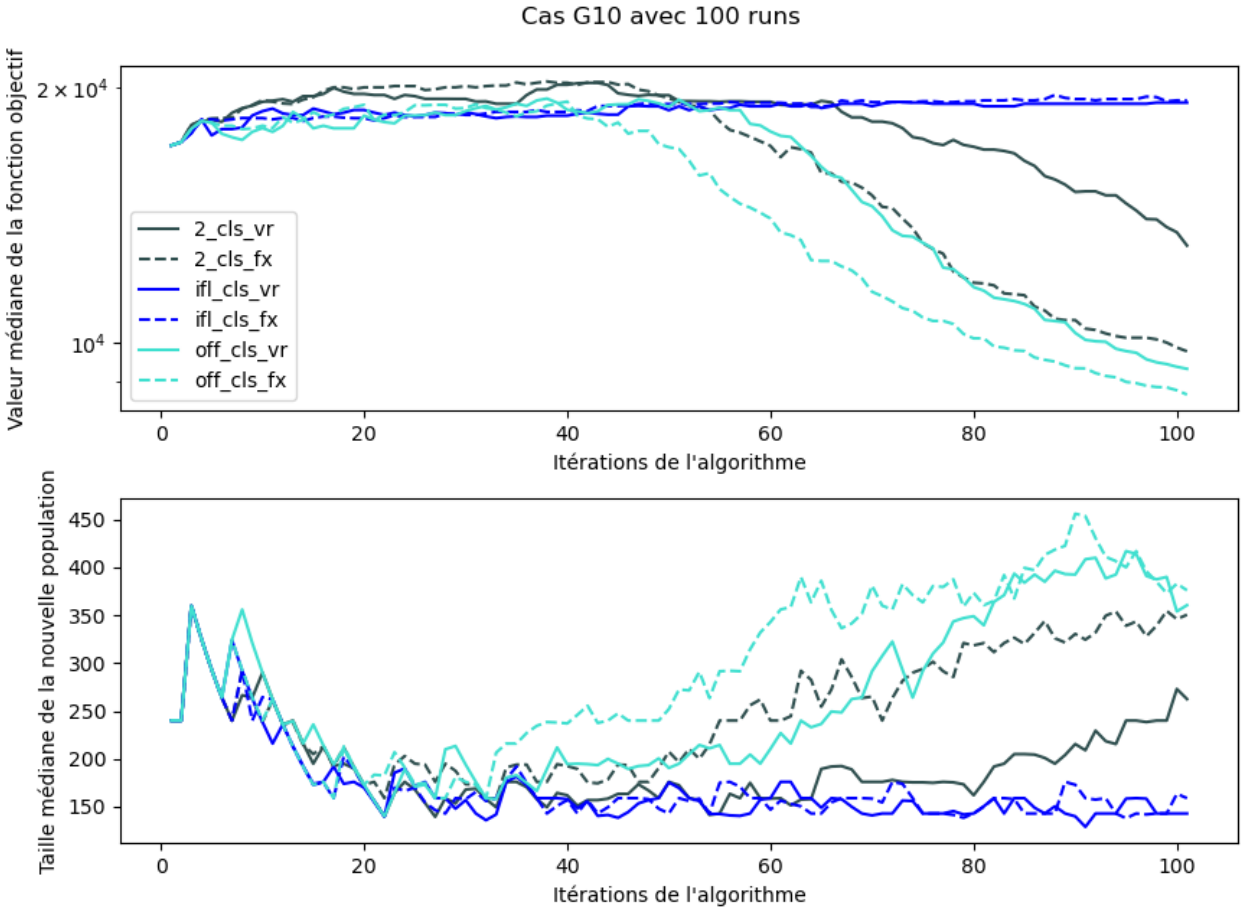


FIGURE 5.3: Graphiques présentant, pour le cas-test G10, l'évolution de la valeur médiane de la fonction objectif (haut) et celle de la taille médiane de la population (bas), toutes les deux par rapport au nombre d'itérations effectuées par les algorithmes concernés.

Chapitre 6

Résultats et comparaisons des versions de la OFSR

Pour rappel, nous avons présenté, dans la Section 3.3, différentes stratégies qui ont mené à la mise en place de plusieurs versions de la OFSR. Pour pouvoir comparer et analyser la performance de ces dernières, nous avons décidé de les séparer en différentes familles qui s'articuleront autour des stratégies concernant la suppression d'individus. En particulier, nous travaillerons sur quatre familles nommées

- *ErasingByProbWithClassic* (F_1), pour les versions supprimant des individus via leur probabilité de survie définie par (3.5) ;
- *ErasingByProbWithMean* (F_2), pour les versions supprimant des individus via leur probabilité de survie définie par (3.7) où la moyenne des *GO* est utilisée ;
- *ErasingByProbWithMedian* (F_3), pour les versions supprimant des individus via leur probabilité de survie définie par (3.7) où la médiane des *GO* est utilisée ;
- *ErasingByGlobalObjective* (F_4), pour les versions supprimant des individus via les valeurs de leur *GO*.

Pour chacune de ces familles, nous analyserons les courbes et valeurs des profils de performance correspondant aux différentes versions qui les composent. Leurs interprétations nous permettront de déterminer quelle sera la version, par famille, que nous retiendrons pour la suite. Par après, nous comparerons la performance des versions sélectionnées afin de déterminer quelles seront les variantes de la OFSR avec lesquelles nous travaillerons dans la suite de ce mémoire. Tout au long de ce chapitre, nous chercherons également à comprendre les raisons qui expliqueraient les différences de performance entre chaque version. Notons que la tolérance ε_f, introduite dans (5.2) au début du chapitre précédent, que nous avons fixée pour chaque cas-test, ne sera pas modifiée lorsque nous changerons de famille. En particulier, le Tableau 6.1 renseigne les valeurs qui ont été utilisées. Tous ces choix ont été déterminés afin de discriminer au mieux les différentes versions considérées tout en s'assurant que tous les cas-tests puissent être résolus par au moins une version, pas nécessairement commune.

Cas-test	Ackley_10	G2_plog_10	G7	G10	Rastrigin_10	Rosenbrock_10	Schwefel_10
Valeur de ε_f	5e-4	3.3e-1	6e+0	3e+3	3e+1	7e+0	2.1e+3

Tableau 6.1: Présentation des valeurs de la tolérance absolue utilisée pour chaque cas-test lors de la comparaison des versions de la OFSR par famille.

6.1 Analyse de la performance globale

6.1.1 Famille *ErasingByProbWithClassic* (F_1)

Pour cette famille de versions, la présentation et l'interprétation du graphique de leur profil de performance ont déjà été faites dans la Section 5.1 via la Figure 5.1 et le Tableau 5.1. Pour résumer ce qui a été écrit à ce sujet, nous pouvons y observer que la version la plus performante est la off_cls_vr. En particulier, lorsque le budget accordé est minimal, pour τ égal à un, son taux de cas-tests résolus est supérieur à celui des autres versions. De plus, lorsque le budget est augmenté, nous pouvons remarquer sur cette Figure 5.1 que les profils de performance se stabilisent sur quatre niveaux différents et que la version off_cls_vr se trouve sur le plus haut niveau.

Pour la suite, nous déciderons donc de considérer la version off_cls_vr comme étant la plus performante de la famille F_1. Ce choix est motivé pour deux raisons, tout d'abord, cette version fait partie de celles possédant la valeur la plus haute pour leur profil de performance lorsque τ augmente. Toutefois, cet argument vaut aussi pour la version off_cls_fx mais ce qui les différencie constitue la raison pour laquelle nous n'avons pas conservé cette version-ci. En effet, lorsque nous nous intéressons aux profils de performance pour un budget limité au minimum possible, c'est la version **off_cls_vr** qui possède la plus grande, et donc la meilleure valeur.

6.1.2 Famille *ErasingByProbWithMean* (F_2)

Pour cette famille de versions, le graphique des profils de performance correspondants est visible à la Figure 6.1. Notons que l'information visuelle tirée de ce dernier est complétée par les résultats quantitatifs présentés dans le Tableau 6.2. Nous pouvons observer sur cette figure, lorsque τ augmente, que trois versions parviennent à résoudre la totalité des cas-tests considérés. Autrement dit, les versions 2_mean_vr, off_mean_fx et off_mean_vr sont les plus robustes de cette famille. Comme nous l'avons fait précédemment, nous nous intéresserons également à l'efficacité des différentes versions en observant les profils de performance lorsque le paramètre τ vaut un. Dans ce cas-ci, c'est la version off_mean_vr qui se démarque de toutes les autres puisque cette dernière présente un taux de réussite égal à 0.71 qui est plus de cinq fois supérieur à la valeur du second meilleur taux obtenu (0.14). Nous pouvons en conclure que si la version **off_mean_vr** n'est pas la seule qui se distingue en terme de robustesse, elle est en revanche celle qui est de loin la plus efficace. Voilà pourquoi cette dernière sera retenue pour la suite de ce mémoire.

version (acronyme)	$\rho_a(1)$	$\rho_a(\tau_{max})$	best τ
GA_Both_Mean (2_mn_vr)	0.00	1.00	1.70
GA_Both_Mean_fx (2_mn_fx)	0.00	0.86	1.40
GA_Infill_Mean (ifl_mn_vr)	0.14	0.14	1.00
GA_Infill_Mean_fx (ifl_mn_fx)	0.00	0.43	2.10
GA_Offspring_Mean (off_mn_vr)	0.71	1.00	1.90
GA_Offspring_Mean_fx (off_mn_fx)	0.14	1.00	1.50

Tableau 6.2: Résultats quantitatifs des profils de performance correspondant à la famille F_2 et illustrés par la Figure 6.1.

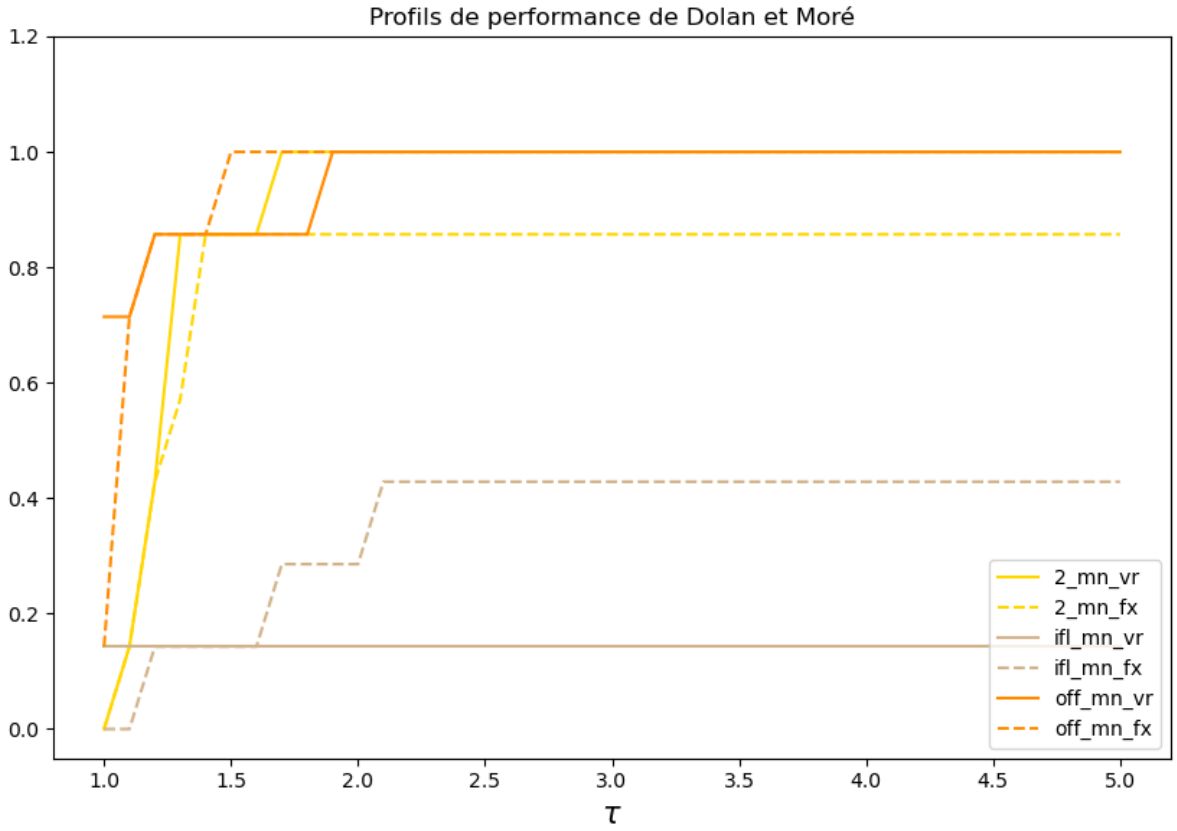


FIGURE 6.1: Graphique des profils de performance obtenus pour les versions de la OFSR appartenant à la famille F_2 sur l'ensemble de cas-tests défini dans le Chapitre 4.

6.1.3 Famille *ErasingByProbWithMedian* (F_3)

À nouveau, nous analyserons les profils de performance des versions de cette famille en utilisant leur représentation graphique à la Figure 6.2 et leurs résultats quantitatifs donnés par le Tableau 6.3. En particulier, nous pouvons observer que seules deux versions présentent un profil de performance atteignant sa valeur maximale. Il s'agit des versions off_median_fx et off_median_vr. De plus, le graphique de la Figure 6.2 nous indique que ces deux versions présentent des profils de performance avec un comportement majoritairement similaire puisque leurs courbes finissent par rapidement se confondre. Plus précisément, ces deux versions se distinguent seulement pour des petites valeurs de τ, correspondant à une petite proportion du budget minimal accordé. De plus, nous pouvons voir que pour ces valeurs, le profil de performance de la version off_median_vr est toujours le plus élevé. En particulier, le Tableau 6.3 indique que ce dernier vaut pratiquement deux fois celui de la version off_median_fx lorsque la valeur de τ est fixée à un. Cela signifie que même si nous ne pouvons pas distinguer ces versions en terme de robustesse, nous pouvons affirmer que la version **off_median_vr** est la plus efficace. Dès lors, cette dernière sera celle que nous désignerons comme la version la plus performante pour la famille F_3.

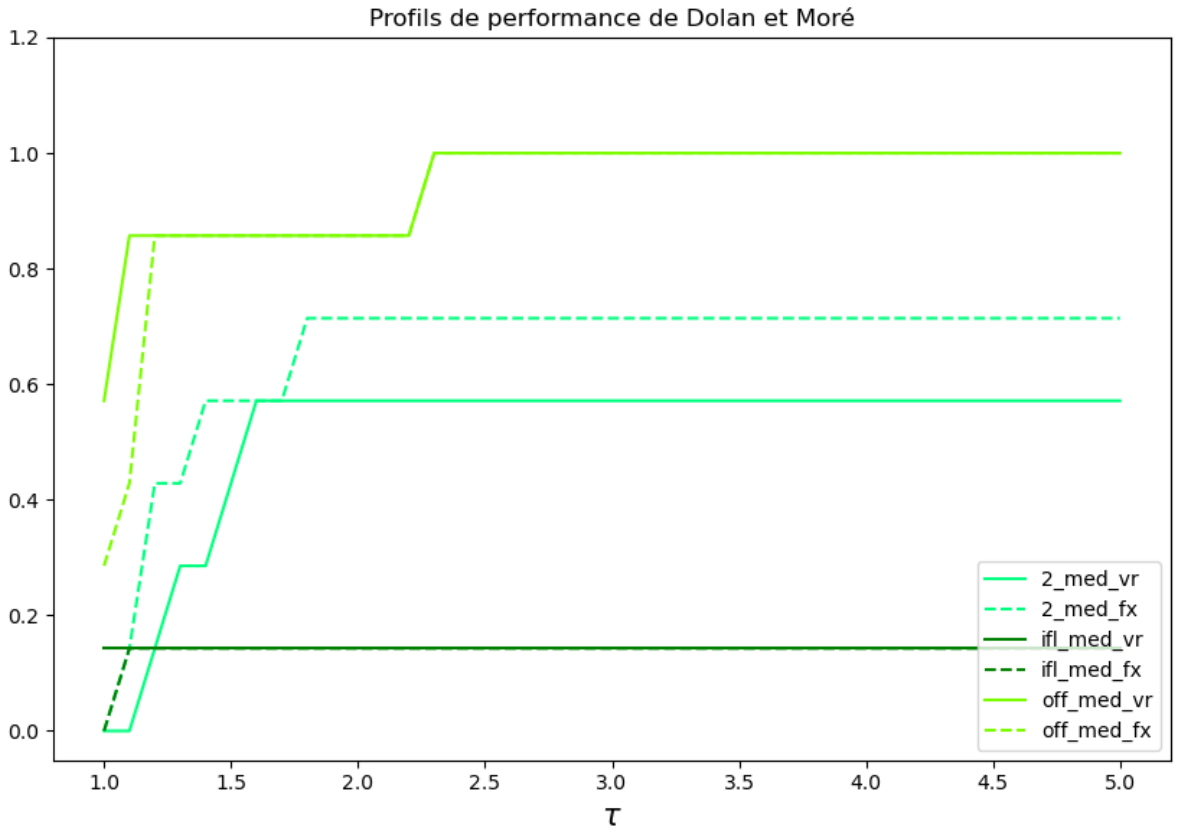


FIGURE 6.2: Graphique des profils de performance obtenus pour les versions de la OFSR appartenant à la famille F_3 sur l'ensemble de cas-tests défini dans le Chapitre 4.

version (acronyme)	$\rho_a(1)$	$\rho_a(\tau_{max})$	best τ
GA_Both_Median (2_med_vr)	0.00	0.57	1.60
GA_Both_Median_fx (2_med_fx)	0.00	0.71	1.80
GA_Infill_Median (ifl_med_vr)	0.14	0.14	1.00
GA_Infill_Median_fx (ifl_med_fx)	0.00	0.14	1.10
GA_Offspring_Median (off_med_vr)	0.57	1.00	2.30
GA_Offspring_Median_fx (off_med_fx)	0.29	1.00	2.30

Tableau 6.3: Résultats quantitatifs des profils de performance correspondant à la famille F_3 et illustrés par la Figure 6.2.

6.1.4 Famille *ErasingByGlobalObjective* (F_4)

Comme nous l'avons fait pour les précédentes familles nous utiliserons les informations graphiques et quantitatives sur les profils de performance pour déterminer quelle version sera retenue pour être comparée avec celles mises en évidence dans les sections précédentes. Ces informations sont visibles à la Figure 6.3 et au Tableau 6.4. Nous pouvons observer que la version la plus performante de cette famille est la off_go. En effet, cette dernière arrive à résoudre la totalité des cas-tests quand son budget est augmenté. De plus, lorsqu'il est limité au minimum, la version off_go présente un profil de performance avec une valeur égale à 0.86 alors que le second meilleur profil possède une valeur égale à 0.14, soit environ six fois plus petite. Il semble alors pertinent de considérer la version **off_go** comme la plus performante issue de la famille F_4.

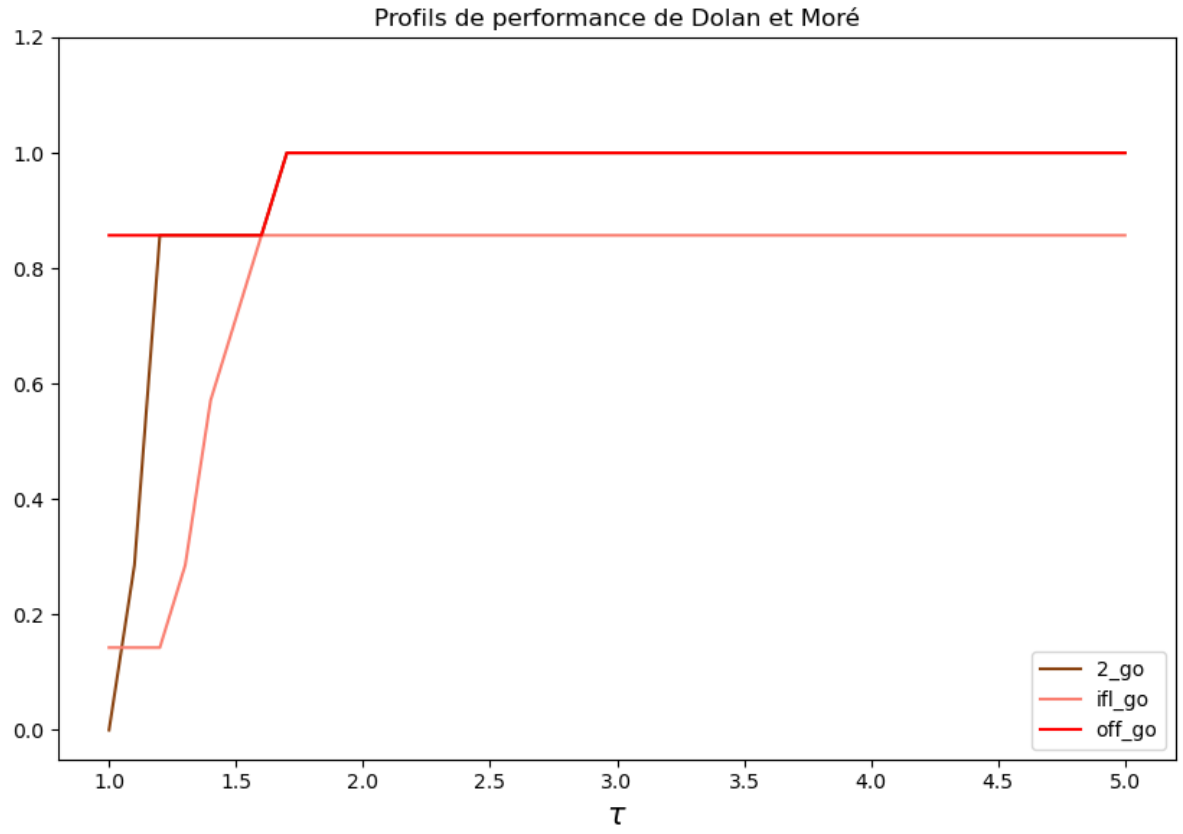


FIGURE 6.3: Graphique des profils de performance obtenus pour les versions de la OFSR appartenant à la famille F_4 sur l'ensemble de cas-tests défini dans le Chapitre 4.

version (acronyme)	$\rho_a(1)$	$\rho_a(\tau_{max})$	best τ
GA_Both_GO (2_go)	0.00	1.00	1.70
GA_Infill_GO (ifl_go)	0.14	0.86	1.60
GA_Offspring_GO (off_go)	0.86	1.00	1.70

Tableau 6.4: Résultats quantitatifs des profils de performance correspondant à la famille F_4 et illustrés par la Figure 6.3.

6.2 Questionnement sur les résultats obtenus

Si nous reprenons l'ensemble des versions que nous avons sélectionnées dans la section précédente, il s'avère qu'elles possèdent toutes un élément en commun. En effet, elles utilisent la même stratégie lorsque la nouvelle population dans l'algorithme MINAMO doit être augmentée. Cette dernière est nommée *Offspring* (*off*) et consiste, pour rappel, à générer des enfants supplémentaires à partir de la population courante pour compléter celle qui sera évaluée à la génération suivante. De plus, lorsque la stratégie de suppression d'individus utilise la notion de probabilité de survie, la manière dont le nombre aléatoire, utilisé dans (3.6) pour gérer cette suppression, est tiré correspond pour chacune des versions sélectionnées. Plus précisément, ce dernier sera toujours généré pour chaque individu de la population concernée et nous représenterons ce cas par le terme *vr*.

En ce qui concerne la stratégie lors de l'ajout d'individus, les profils de performance relatifs à chaque famille montrent une séparation assez claire qui correspond aux trois stratégies utilisées. Par exemple, les Figures 5.1 et 6.2 permettent de voir explicitement la séparation entre ces stratégies. De manière générale, parmi tous les profils que nous avons obtenus, nous pouvons observer que la stratégie *Infill* est la moins performante, suivie par la *Both* pour terminer avec la *Offspring* qui les surpasse. Cela pourrait s'expliquer par le fait que la stratégie *Infill* ajoute des individus uniquement de manière aléatoire et ce, sans tenir compte de l'état de la recherche. Par conséquent, cela pourrait entrainer une exploration trop importante de l'espace de recherche qui empêcherait les versions usant de cette stratégie de s'approcher de l'optimum global recherché pour leurs différents cas-tests. A contrario, la stratégie *Offspring* utilise la population courante pour générer de nouveaux enfants afin d'augmenter la taille de la population à la génération suivante. Dès lors, les versions utilisant cette stratégie seraient plus aptes à converger vers l'optimum global recherché puisque l'ajout de nouveaux individus tiendrait compte de l'état de la recherche. En ce qui concerne la stratégie *Both*, le fait qu'elle utilise une combinaison des deux précédentes expliquerait pourquoi elle est meilleure que la stratégie *Infill* mais moins bonne que la *Offspring*.

Par rapport aux versions utilisant la notion de probabilité de survie, la qualité entre les choix du tirage du nombre aléatoire utilisé n'est pas aussi facilement déterminable. Prenons le graphique à la Figure 6.2 par exemple. Sur ce dernier, le tirage le plus optimal varie en fonction de la stratégie utilisée pour ajouter des individus. En effet, les versions utilisant la *Infill* montrent une performance fort semblable peu importe le choix du tirage. En particulier, les valeurs du Tableau 6.3 nous indiquent que la seule différence se trouve lorsque le paramètre τ est inférieur à 1.10. Cette différence indique que pour ces valeurs de τ, le tirage variable (*vr*) est le plus performant puisque la courbe du profil correspondant est supérieure à celle du tirage fixe (*fx*). Cependant, si nous regardons les courbes liées à la stratégie *Both*, c'est celle de la version utilisant le tirage fixe qui est la plus performante. Il semble donc que le tirage le plus optimal dépende de la stratégie utilisée pour l'ajout d'individus. En particulier, pour la *Offspring*, c'est le tirage variable qui semble être celui à privilégier lorsque la suppression d'individus dépend d'une probabilité de survie.

6.3 Meilleures versions de la OFSR

En plus des quatre versions mises en avant dans la Section 6.1, nous considérerons également celles faisant partie de la famille F_4 pour déterminer les deux meilleures versions de la OFSR. Nous avons fait ce choix car, lors du traitement des résultats obtenus pour les versions de cette famille, ces dernières nous semblaient présenter de meilleurs résultats par rapport à celles des autres familles. En résumé, les versions finales que nous avons considérées pour déterminer les meilleures variantes de la OFSR sont

- off_cls_vr ;
- off_mean_vr ;
- off_median_vr ;
- 2_go ;
- ifl_go :
- off_go.

Pour ces dernières, nous avons calculé les profils de performance correspondants dont les courbes, complétées par le Tableau 6.5, se trouvent à la Figure 6.4. Nous pouvons y observer que toutes les versions considérées, sauf la ifl_go, arrivent à atteindre la valeur maximale pour leur profil de performance. Dès lors, cette dernière ne sera plus prise en compte puisqu'elle n'est pas capable de résoudre la totalité des cas-tests pour le budget considéré. Pour les versions restantes, leur distinction se fera lorsque le paramètre τ aura des valeurs proches de un puisqu'elles arrivent toutes au taux maximal de cas-tests résolus. En particulier, nous pouvons voir que le profil de la version off_go atteint une valeur de 0.86 lorsque $\tau = 1$. Autrement dit, il s'agit de la version qui, pour une majorité écrasante des cas-tests, arrive à les résoudre avec le budget le plus petit. C'est donc la version la plus efficace parmi toutes celles considérées.

Parmi les autres versions qui arrivent à résoudre la totalité des cas-tests lorsque τ augmente, le Tableau 6.5 nous indique que c'est la 2_go qui est la plus efficace. En effet, sa valeur pour la colonne "best τ" est la plus petite parmi celles des versions concernées. Cela signifie que cette dernière demande le plus petit budget pour arriver à résoudre tous les cas-tests et qu'elle est donc bien la plus efficace si nous ne considérons pas la version off_go. Cette interprétation peut également se faire via le graphique à la Figure 6.4 puisque nous pouvons voir sur ce dernier que la courbe du profil de la version **2_go** se confond rapidement avec celle de la version **off_go**. Par conséquent, nous avons décidé de déterminer ces deux versions comme étant celles de la OFSR les plus performantes et nous les comparerons

version (acronyme)	$\rho_a(1)$	$\rho_a(\tau_{max})$	best τ
GA_Both_GO (2_go)	0.00	1.00	1.70
GA_Infill_GO (ifl_go)	0.14	0.86	1.60
GA_Offspring_Classic (off_cls_vr)	0.00	1.00	2.60
GA_Offspring_GO (off_go)	0.86	1.00	1.70
GA_Offspring_Mean (off_mn_vr)	0.00	1.00	2.60
GA_Offspring_Median (off_med_vr)	0.00	1.00	2.40

Tableau 6.5: Résultats quantitatifs des profils de performance correspondant à ceux illustrés dans la Figure 6.4.

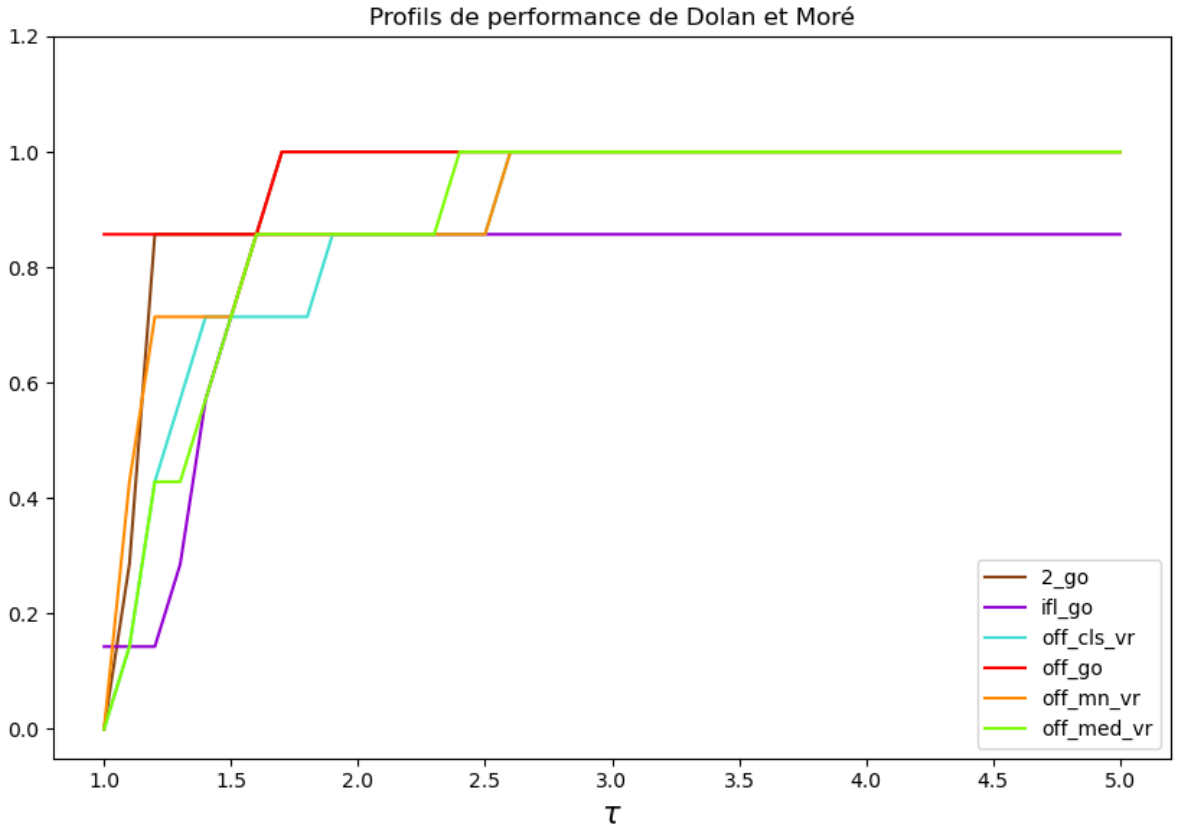


FIGURE 6.4: Graphique des profils de performances obtenus pour les versions de la OFSR sélectionnées à partir des familles F_1, F_2, F_3 et F_4.

dans la suite de ce mémoire à la version initiale de la OFSR et aussi à celle par défaut de MINAMO. Pour compléter notre analyse, déjà claire à ce stade sur les meilleures versions, vous trouverez également en Annexe A les outils qui ont été présentés dans la Section 5.2. Ces derniers permettent d'obtenir des informations plus détaillées pour chaque cas-test et confirment les différentes conclusions que nous avons mises en avant dans cette section.

Si l'analyse des profils de performance dans les différentes familles de versions nous a permis de réaliser que la stratégie d'ajout d'individus *Offspring* était la plus performante. Les résultats pour l'ensemble final des versions nous montrent que la stratégie de suppression d'individus qui se combine le mieux avec cette dernière est la *GO*. La raison derrière cela pourrait être le caractère aléatoire des stratégies de suppression utilisant une probabilité de survie. En effet, pour chaque individu de la population à réduire, plus la valeur de son *Global Objective* est meilleure, plus il aura de chance de survivre car sa probabilité de survie sera proche de un. Cependant, il est possible que le nombre aléatoire utilisé pour déterminer la survie de chaque individu soit encore plus proche de un. Auquel cas, mêmes des individus avec un bon *Global Objective* seraient supprimés. Le contraire est également possible, si ce nombre est proche de zéro, alors de mauvais individus pourraient survivre. Du côté de la stratégie *GO*, la méthode employée est plus radicale puisqu'elle supprime directement les individus avec le moins bon *Global Objective*. De cette manière, nous opérons une sélection supplémentaire qui pourrait s'apparenter à de l'élitisme puisque les individus supprimés

sont obligatoirement les moins performants. Cela pourrait expliquer pourquoi c'est cette stratégie plus extrême qui s'avère être la plus performante dans les résultats que nous avons obtenus.

Nous connaissons maintenant les versions de la *One Fifth Success Rule* qui seront comparées avec celle par défaut de MINAMO et également avec la version initiale utilisant la OFSR, présentée dans la Section 3.2. En effet, parmi toutes les variantes de la OFSR que nous venons de comparer, ce sont les versions off_go et 2_go qui se sont avérées être les plus performantes. Le chapitre qui va suivre nous permettra de visualiser l'impact de la gestion dynamique de la taille de la population sur l'algorithme génétique présent dans MINAMO.

Chapitre 7

Comparaison avec la version par défaut de Minamo

Ce chapitre va nous permettre de confronter la version par défaut de l'algorithme génétique de Minamo avec celles modifiées selon la règle de la *One Fifth Success Rule* pour la gestion de la taille de la population par génération. Notons que dans la suite, nous ne préciserons plus qu'il s'agit de l'algorithme génétique de Minamo. En particulier, nous considérerons la version initiale de la OFSR présentée dans la Section 3.2 dont l'implémentation dans Minamo est décrite par l'Algorithme 2. Parmi les nombreuses variantes définies dans la Section 3.3, l'analyse du chapitre précédent nous a permis de sélectionner celles utilisant la stratégie *Offspring* et *Both* pour gérer l'ajout d'individus dans la population. En ce qui concerne la réductions de cette dernière, les deux variantes retenues utilisent la stratégie *GO*. Pour pouvoir distinguer ces différentes versions, nous appellerons celle initiale GA_OneFifth (onefifth) et ses deux variantes seront nommées GA_Offspring_GO (off_go) et GA_Both_GO (2_go).

Rappelons que la version par défaut de Minamo se distingue de ces versions modifiées par le fait que la taille de la population reste constante pendant toute son exécution. De plus, lors de la mise en place des versions onefifth, off_go et 2_go, nous avons fixé différentes bornes ainsi qu'une valeur initiale pour déterminer la taille de la population. C'est pour cela que nous avons exécuté la version par défaut de Minamo en modifiant le nombre d'individus que la population devait contenir. Plus précisément, ces différentes versions de Minamo seront nommées

- GA_10,
- GA_30,
- GA_50,
- GA_100,

où le nombre utilisé indique le coefficient par lequel nous multiplions la dimension du problème d'optimisation pour obtenir le nombre d'individus dans la population. Par exemple, la version GA_30 utilisera un nombre d'individus dans la population égal à trente fois la dimension du cas-test considéré.

Le choix de ces différents nombres n'est pas fait au hasard. En effet, le premier correspond à la valeur du coefficient utilisé dans l'équation (3.2) pour définir la borne inférieur de la taille de la population dans les versions utilisant la OFSR. Dans la même logique, le

Cas-test	Ackley_10	G2_plog_10	G7	G10	Rastrigin_10	Rosenbrock_10	Schwefel_10
Valeur de ε_f	1e-4	3.2e-1	4e+0	1.5e+3	1e+0	6.9e+0	1.95e+3

Tableau 7.1: Présentation des valeurs de la tolérance absolue utilisée pour chaque cas-test lors de la comparaison des meilleures versions de la OFSR à celles par défaut de MINAMO.

second correspond au coefficient permettant de définir la taille initiale et le dernier à celui utilisé pour le calcul de la borne supérieure. La seule valeur qui n'a pas de lien directe avec la OFSR est la troisième. Nous avons décidé de la considérer pour servir de valeur intermédiaire entre le coefficient de la taille initiale et celui de la borne supérieure. Ces différents nombres pour la version par défaut de MINAMO vont nous permettre de la comparer de manière pertinente avec les deux versions utilisant la OFSR. En effet, comme ces dernières modifient la taille de la population pendant leur exécution, nous avons voulu que les versions par défaut de MINAMO respectent l'amplitude que pouvait prendre cette taille. Avant de passer à la comparaison de la performance des versions par défauts de MINAMO avec celles utilisant la *One Fifth Success Rule*, notons que nous avons décidé de fixer une tolérance ε_f pour chaque cas-test plus stricte que celle utilisée au chapitre précédent. En particulier, les différentes valeurs pour cette tolérance sont données dans la Tableau 7.1. Ces choix ont été faits car nous souhaitions augmenter l'exigence demandée lors de la comparaison des performances entre les versions par défaut de MINAMO et celles utilisant le OFSR.

Pour pouvoir effectuer cette comparaison, nous avons à nouveau utilisé les profils de performance décrits dans la Section 5.1. Le graphique de ces derniers est présenté dans la Figure 7.1 et le Tableau 7.2, contenant des données quantitatives sur ces profils, complète ces résultats graphiques. Commençons par analyser les informations fournies par le Tableau 7.2. Ces dernières nous indiquent que la version la plus performante pour τ égal à un est la GA_10. Autrement dit, il s'agit de celle qui demande un budget minimal pour résoudre le plus de cas-tests. En ce qui concerne les autres versions par défaut de MINAMO, nous pouvons observer que plus le coefficient pour la taille de la population est important, moins la version en question est performante pour une valeur de τ proche de un. Cela pourrait s'expliquer par le fait qu'un trop grand nombre d'individus dans la population entrainerait une trop grande exploration. Dès lors, il serait difficile de converger rapidement vers la solution globale. En revanche, lorsque le budget autorisé est maximal, la version GA_10 se fait dépasser par toutes les autres versions sauf la GA_100 qui s'avère être celle avec le profil de performance final le moins élevé. Ce résultat illustre bien la nécessité d'un équilibre entre une petite taille de population pour exploiter l'espace de recherche et une grande taille pour

version (acronyme)	$\rho_a(1)$	$\rho_a(\tau_{max})$	best τ
GA_10 (GA_10)	0.43	0.43	1.00
GA_30 (GA_30)	0.29	0.57	2.90
GA_50 (GA_50)	0.00	0.71	5.00
GA_100 (GA_100)	0.00	0.29	3.80
GA_OneFifth (onefifth)	0.14	0.71	4.20
GA_Offspring_GO (off_go)	0.14	1.00	4.10
GA_Both_GO (2_go)	0.00	1.00	4.70

Tableau 7.2: Résultats quantitatifs des profils de performance correspondant à ceux illustrés dans la Figure 7.1.

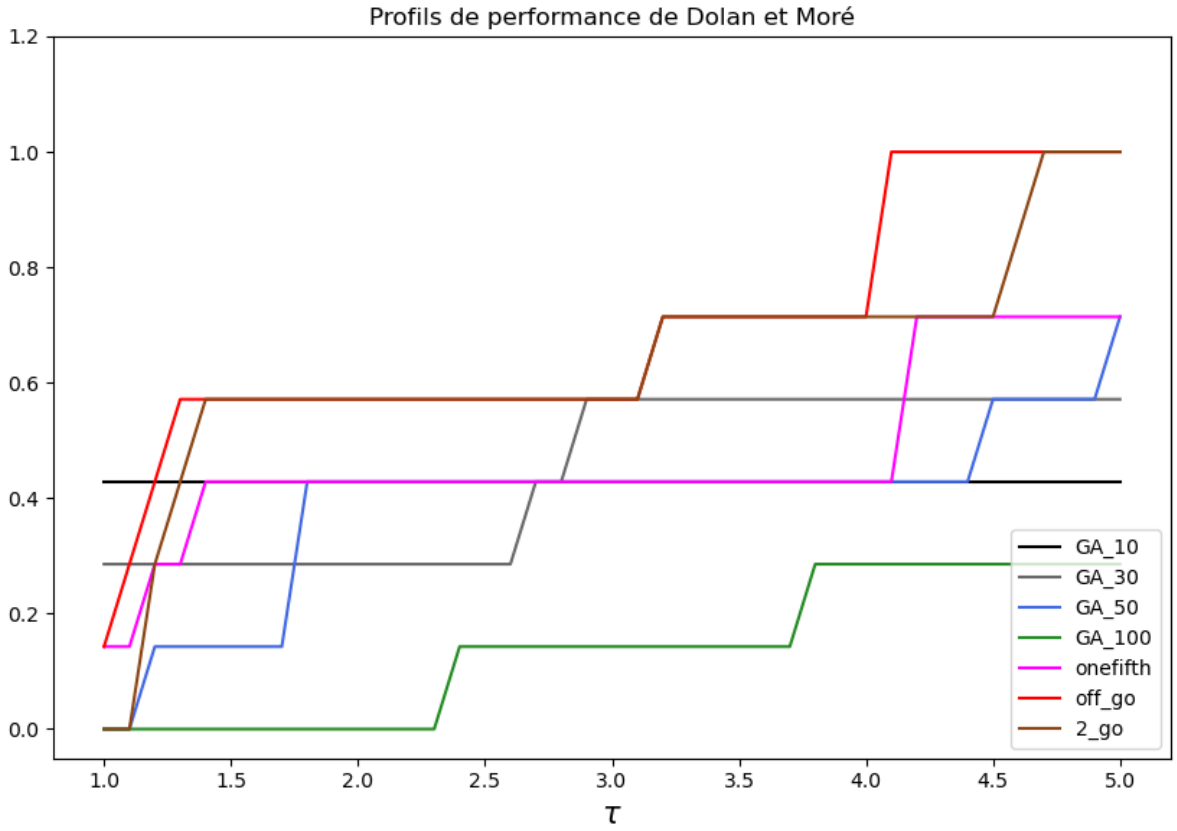


FIGURE 7.1: Graphique des profils de performances obtenus pour les versions par défaut de MINAMO et celles utilisant la OFSR.

favoriser l'exploration. En effet, la version GA_10 illustre une exploitation trop poussée qui suffit pour certains cas-tests mais qui lui porte préjudice pour d'autres. Cela se voit par le fait que son profil de performance reste constant malgré le fait que la valeur du paramètre τ augmente. Pour la version GA_100, le fait d'avoir un nombre trop important d'individus dans sa population l'empêche de converger vers la solution des différents cas-tests considérés. Autrement dit, l'exploration est ici trop importante par rapport à l'exploitation. En plus de cette illustration sur l'équilibre nécessaire entre l'exploration et l'exploitation, le Tableau 7.2 nous indique que les versions les plus robustes parmi toutes celles considérées sont la off_go et la 2_go. En effet, ce sont les seules dont le profil de performance atteint sa valeur maximale de un. En outre, le "best τ" pour ces deux versions dans le Tableau 7.2 nous indique que la version off_go est la plus efficace car elle demande la plus petite proportion du budget minimal requis pour résoudre l'ensemble des cas-test considérés.

Pour compléter notre interprétation, la Figure 7.1 nous permet de voir que même si la courbe du profil de performance de off_go n'est pas la plus haute pour τ égal à un, cette dernière surclasse rapidement toutes celles qui la surpassaient et ne se fait plus dépasser par la suite. Nous pouvons également ajouter qu'il s'agit de la version qui arrive à dépasser la moitié des cas-tests résolus pour la plus petite valeur de τ. Dès lors, il semble cohérent de considérer cette version comme, globalement, la plus performante. De plus, nous pouvons ajouter que les versions onefifth et GA_50 sont celles dont les profils de performances se

rapprochent le plus de un lorsque τ augmente. En particulier, leurs valeurs finales sont égales. Cependant, le profil de la version initiale de la OFSR atteint sa valeur maximale avant la version GA_50, ce qui indique que c'est la première qui est la plus efficace puisqu'elle demande un plus petit budget pour arriver à ce taux de cas-tests résolus. Pour résumer notre analyse, nous pouvons observer que les versions les plus robustes sont

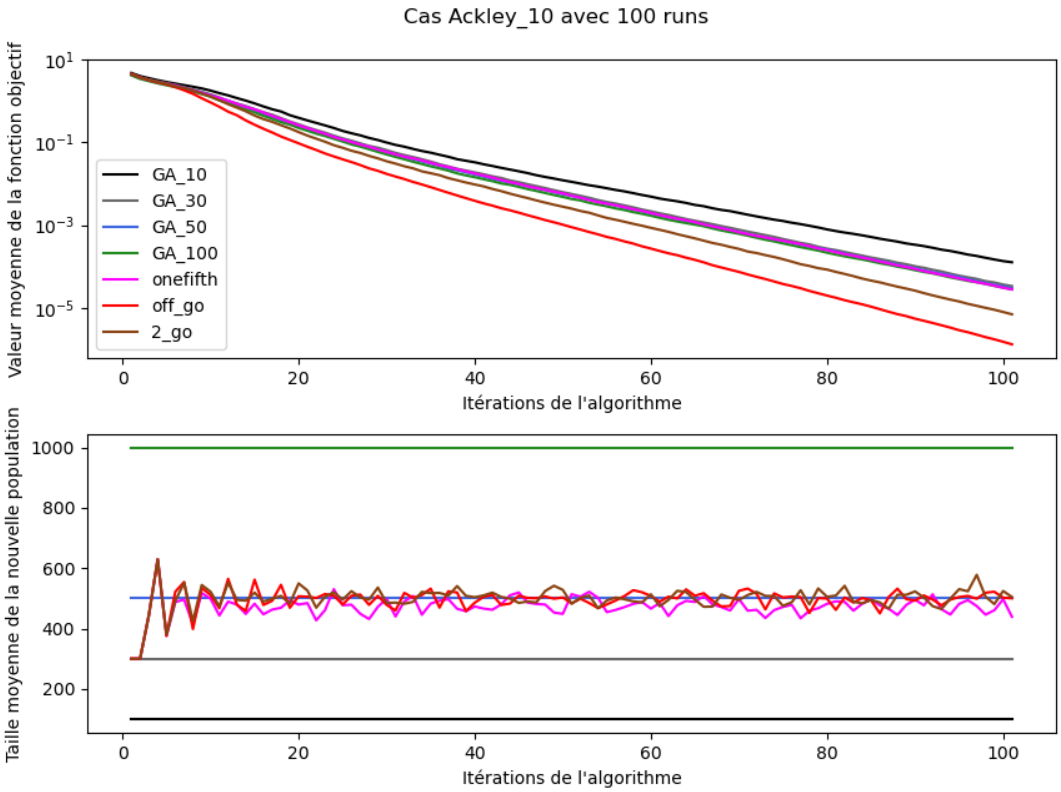
- off_go,
- 2_go,
- onefifth,
- GA_50,

où celle qui se démarque le plus est la off_go car son profil atteint sa valeur maximale pour la plus petite valeur du paramètre τ.

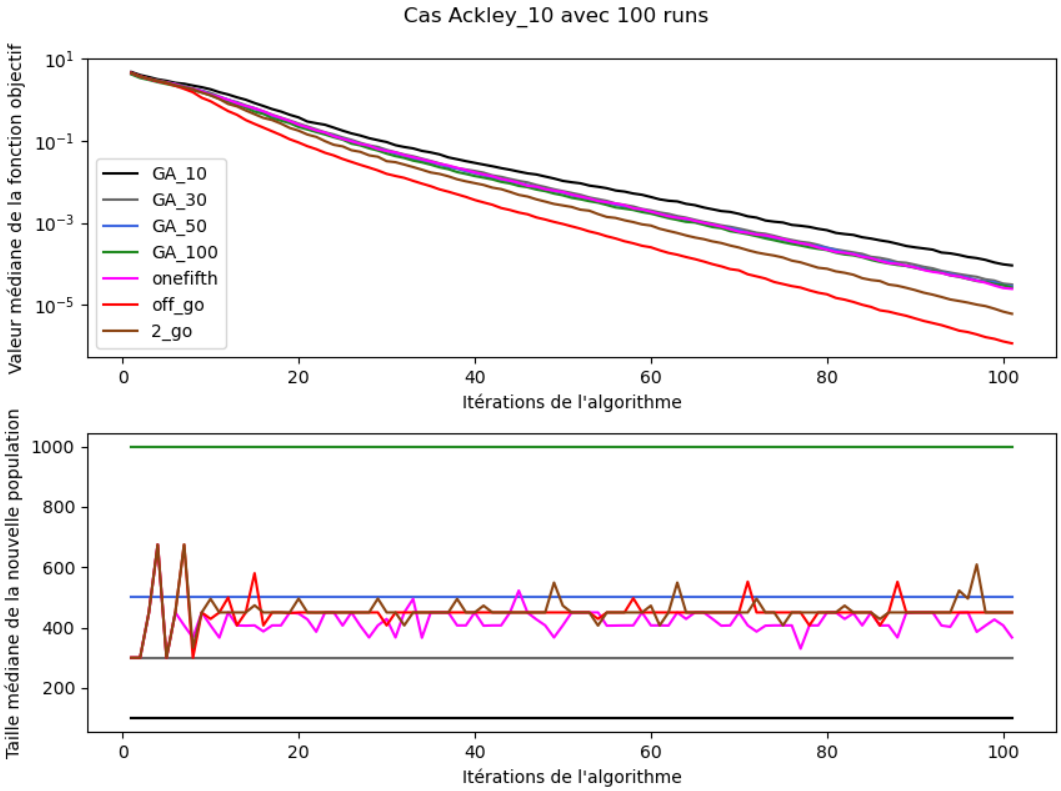
Pour comprendre les raisons derrière cela, nous avons pu remarquer que la taille de la population des versions utilisant la *One Fifth Success Rule* avait tendance à évoluer entre l'intervalle des tailles de population pour les versions par défaut de Minamo GA_30 et GA_50. En effet, la Figure 7.2 nous permet de voir l'évolution du nombre moyen et médian d'individus dans la population en fonction des itérations pour le cas-test Ackley_10. Ces graphiques font partie des résultats présentés dans la Section 5.2. En particulier, nous pouvons observer sur la Figure 7.2a que la taille moyenne de la population pour les versions utilisant la OFSR oscillent au niveau de celle correspondant à la version GA_50. En ce qui concerne l'évolution de la taille médiane, la Figure 7.2b nous montre que cette dernière oscille également autour d'une valeur proche de la taille de la population pour la version GA_50. Afin de se convaincre de ce comportement, nous avons représenté à la Figure 7.3 les graphiques similaires pour le cas-test G10. Ces derniers nous indiquent que le nombre d'individus moyen et médian de la population des versions 2_go, off_go et onefifth oscille également autour d'une valeur se trouvant entre la taille de la population pour la version GA_30 et la GA_50. Ces comportements pourraient signifier que les versions de Minamo utilisant la OFSR tentent naturellement d'approcher la taille de la population la plus optimale pour résoudre chaque cas-test. Dès lors, comme nous avons pu voir au travers des Figures 7.2 et 7.3, il semblerait que cette valeur optimale se trouve entre trente et cinquante fois la dimension du cas-test considéré. Ces observations ont pu être observées dans la majorité des cas-tests considérés pour lesquels vous trouverez, en Annexe B, l'ensemble des résultats venant des outils présentés à la Section 5.2. Cela expliquerait le fait que les versions par défaut de Minamo les plus robustes soient justement celles utilisant ces valeurs pour définir la taille de leur population respective. Dès lors, comme les versions 2_go, off_go et onefifth ont la capacité d'adapter le nombre d'individus dans leur population, cela justifierait le fait que ces dernières soient les plus robustes puisqu'elles ont la possibilité d'atteindre la taille la plus optimale.

En ce qui concerne les différents résultats pour les versions utilisant la *One Fifth Success Rule*, ces derniers ont déjà été expliqués, dans la Section 6.2, pour les versions off_go et 2_go puisqu'elles n'utilisent pas la même stratégie pour ajouter des individus dans leur population. En ce qui concerne les versions off_go et onefifth, ces dernières ne diffèrent que sur un point. En effet, lorsque la population doit être augmentée, la off_go génère des enfants supplémentaires pour atteindre le nombre d'individus voulu. Cela est similaire avec ce que fait la onefifth puisqu'elle génère directement le bon nombre d'enfants pour atteindre la taille déterminée. En revanche, lorsque la population doit être réduite, la onefifth se contente sim-

plement de générer moins d'enfants pour l'itération suivante alors que la off_go va effectuer une sélection parmi les enfants générés en surnombre et supprimera les moins performants. Comme nous l'avons mentionné au chapitre précédent, cette procédure peut s'apparenter à une sélection élitiste supplémentaire et ce serait cette dernière qui expliquerait la différence de performance entre les versions onefifth et off_go. Pour terminer, les résultats obtenus pour les versions 2_go et onefifth pourraient signifier que, parmi les deux cas possibles lors de la gestion de la population dans l'algorithme génétique de Minamo, ce soit la réduction de la population l'opération la plus déterminante. En effet, bien que la stratégie de la version 2_go, pour augmenter la population, soit moins optimale que celle utilisée par la version onefifth, c'est bien la 2_go qui s'avère être la plus performante. Cela pourrait donc signifier que c'est la stratégie pour diminuer la population qui impacte le plus les performances des versions s'inspirant de la OFSR. De fait, celle utilisée par la version onefifth est moins optimale que celle utilisée par la 2_go et c'est cette dernière version qui présente le meilleur profil de performance si la off_go n'est pas prise en compte.

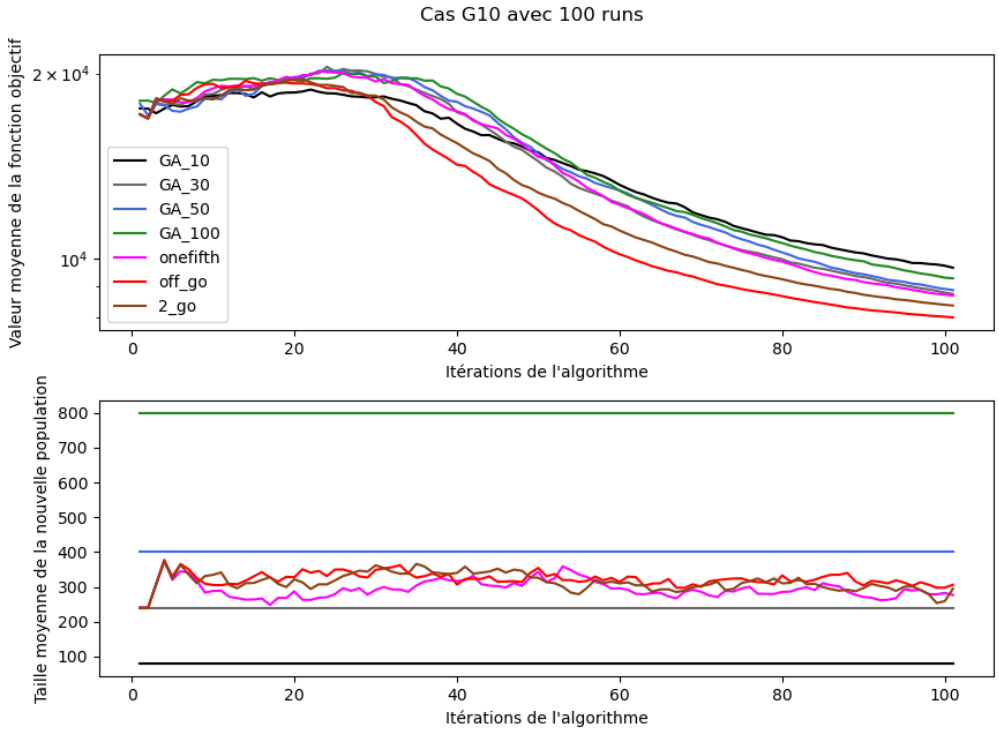


(a) Graphiques des résultats moyens.

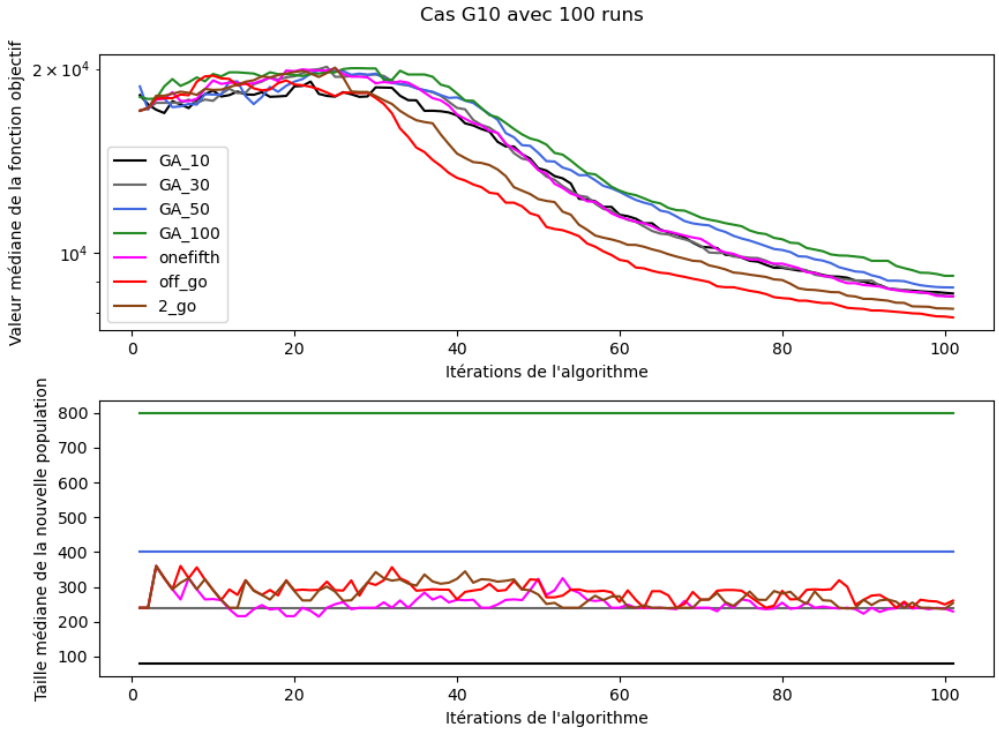


(b) Graphiques des résultats médians.

FIGURE 7.2: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Ackley_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

FIGURE 7.3: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test G10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Conclusions et perspectives

Au cours de ce mémoire, nous avons pu nous rendre compte combien la gestion des paramètres associés aux algorithmes génétiques pouvaient être importante afin de produire des algorithmes performants. En particulier, nous nous sommes intéressés à la gestion de la taille de la population dans ce type d'algorithmes.

Afin de confronter la théorie avec la pratique, nous avons collaboré avec le centre de recherche Cenaero qui nous a permis d'apporter des modifications à l'algorithme génétique présent dans leur logiciel Minamo. Plus précisément, nous avons implémenté, dans cet algorithme, une adaptation de la méthode dynamique adaptative nommée *One Fifth Success Rule* (OFSR) afin de modifier le nombre d'individus de la population lors de l'exécution de ce dernier. Pour résumer son principe décrit au Chapitre 3, cette implémentation va augmenter la taille de la population lorsque la recherche de l'optimum progresse et va la diminuer sinon. En implémentant cette méthode dans Minamo, l'objectif recherché était d'observer comment cette gestion dynamique de la taille de la population allait influencer les résultats de l'algorithme génétique de Minamo.

Lors de ce processus, nous avons mis en place une première version s'inspirant de la OFSR et également un grand nombre de variantes suivant les différentes stratégies utilisées lors de l'implémentation de cette méthode. Afin de pouvoir comparer les performances de ces nouvelles versions entre elles et également avec celle par défaut de Minamo, nous avons utilisé la notion de profil de performance présentée dans [25] pour pouvoir effectuer une comparaison globale sur différents cas-tests. De plus, nous avons aussi mis en place une série de résultats pour pouvoir fournir des outils d'analyse plus détaillés permettant de comparer ces différentes versions sur chaque cas-test séparément.

Une fois ces éléments mis en place, nous avons d'abord comparé la performance des différentes versions issues de la OFSR. L'analyse des résultats obtenus, présentée dans le Chapitre 6, nous a poussés à sélectionner les versions off_go et 2_go comme étant les plus performantes des variantes de la OFSR. Ces dernières utilisent une sélection supplémentaire, semblable à de l'élitisme, lorsque la population doit être diminuée. Dans le cas où le nombre d'individus doit être augmenté, alors la version off_go utilise un procédé déjà implémenté dans l'algorithme génétique de Minamo pour générer des enfants supplémentaires et ce, à partir de la population courante. En ce qui concerne la version 2_go, cette dernière agrandit la taille de la population via une combinaison d'individus générés aléatoirement et d'enfants produits avec le même mécanisme utilisé par la version off_go.

Pour pouvoir analyser les résultats de la version initiale de la OFSR et des deux versions cités précédemment par rapport à ceux de la version par défaut de Minamo, nous avons utilisé différents réglages de cette dernière. En effet, rappelons que l'algorithme génétique

par défaut de Minamo conserve une taille de population constante. Dès lors, nous l'avons exécuté en modifiant la valeur de cette taille pour correspondre aux différentes bornes utilisées par les versions de la OFSR. Ainsi, nous avons travaillé avec des tailles respectant l'amplitude que pouvaient prendre celles calculées via la *One Fifth Success Rule*.

Suite à l'analyse des résultats réalisée dans le Chapitre 7, nous avons pu voir que les versions utilisant la OFSR n'étaient pas celles qui demandaient le plus petit budget pour résoudre le plus grand nombre de cas-test. Pour rappel, le budget est représenté par le nombre médian d'évaluations de la fonction objectif ainsi que des contraintes. En revanche, lorsque nous augmentons le budget disponible, les versions off_go et 2_go sont celles qui se démarquent des autres puisqu'elles sont les seules qui parviennent à résoudre tous les cas-test considérés dans les limites de budget que nous avons imposées. En particulier, la version off_go est celle qui demande le plus petit budget pour atteindre la taux maximal de réussite. En ce qui concerne la version initiale de la OFSR, cette dernière n'arrive pas à résoudre tous les cas-tests mais fait partie des versions qui, pour un large budget, arrivent à résoudre le plus de cas-tests après les versions off_go et 2_go.

Lors de nos analyses, nous avons pu voir que les deux versions par défaut de Minamo qui parvenaient à résoudre le plus de cas-tests étaient celles nommées GA_30 et GA_50. Ces dernières fixent la taille de la population respectivement à trente et cinquante fois la dimension du cas-test considéré. Or, nous avons observé que la taille des versions utilisant la OFSR avait tendance, pour la majorité des cas-tests considérés, à osciller entre les valeurs définies par les versions GA_30 et GA_50. Cela nous a permis de voir que l'utilisation de la *One Fifth Success Rule* permettait à l'algorithme génétique de Minamo de modifier la taille de sa population afin d'atteindre une valeur qui améliore la qualité de sa solution. Pour pouvoir approfondir cette observation, il serait intéressant d'effectuer les analyses faites dans les Chapitres 6 et 7 sur un nombre plus élevé de cas-tests.

De plus, nous devons garder en tête que le travail réalisé dans ce mémoire n'a pas approfondi la manière utilisée pour déterminer si la recherche s'améliorait ou non. Pour rappel, celle avec laquelle nous avons travaillé considère qu'une amélioration était présente si au moins un seul individu de la population courante présentait un *Global Objective* plus performant que celui du meilleur individu de la population précédente. Or, il serait également possible de considérer une amélioration dans un sens plus global. Typiquement, nous pourrions dire que la recherche est améliorée si la moyenne ou la médiane des *GO* de l'ensemble de la population courante est meilleure par rapport à celle de la population précédente. Nous n'avons pas non plus développé l'impact que pouvait avoir la valeur du paramètre α dans la définition de la *One Fifth Success Rule* donnée par (3.2). Ce sont donc des éléments qui mériteraient d'être étudiés afin de voir si la meilleure version de la OFSR retenue ne peut pas être encore améliorée.

Rappelons également que l'utilisation de l'algorithme *SBO* de Minamo, principalement utilisé dans des cas industriels, n'a pas été mis en avant dans ce mémoire. En effet, notre travail s'est concentré sur l'amélioration de l'algorithme génétique, utilisé comme outil par celui travaillant avec des méta-modèles. Dès lors, il serait intéressant d'observer comment ce dernier est impacté lorsque l'adaptation de la *One Fifth Success Rule* est utilisée dans l'algorithme génétique de Minamo. Pour conclure ces différentes perspectives, rappelons que nous avons travaillé sur un seul algorithme génétique dans ce mémoire, celui implémenté

dans le logiciel Minamo de Cenaero. Il serait donc pertinent de mettre en place notre adaptation de la *One Fifth Success Rule* dans d'autres algorithmes afin de voir si nous pouvons retrouver les éléments observés pour celui de Minamo.

Bibliographie

[1] J. Nocedal & S. J. Wright, *Numerical Optimization*, Springer, seconde édition, 683 pages, États-Unis, 2006.

[2] Équipe Minamo, *Workshop Optimization - Session 1 Theoritical Introduction to Optimization*, présentation, Cenaero, 2019.

[3] E. Cuevas & A. Conde & E. Barocio, *Metaheuristics Algorithms in Power Systems*, Springer Nature, Vol. 822, pp. 1-8, Suisse, 2019.

[4] Wikipédia, *Extremum*, `https://fr.wikipedia.org/wiki/Extremum`, consulté le 16 avril 2020.

[5] N. Kokash, *An introduction to heuristic algorithms*, Department of Informatics and Telecommunications, University of Trento, 2005.

[6] Larousse, *Langue Française*, `https://www.larousse.fr/dictionnaires/francais/heuristique/39848?q=heuristique#39767`, consulté le 3 mai 2020.

[7] C. Blum & A. Roli, *Metaheuristic in Combinatorial Optimization : Overview and Conceptual Comparison*, article, ACM Computing Surveys, Vol. 35, No. 3, 2003.

[8] M. Buscema & W.J. Tastle, *Intelligent Data Mining in Law Enforcement Analytics : New Neural Networks Applied to Real Problems*, DOI 10.1007/978-94-007-4914-6_4, © Springer Science+Business Media Dordrecht, pp. 31-48, 2013.

[9] S. McGerty & F. Moisiadis, *Optimised Random Mutations For Evolutionary Algorithms*, International Journal of Artificial Intelligence & Applications (IJAIA), Vol. 5, No. 4, 2014.

[10] A. E. Eiben & R. Hinterding & Z. Michalewicz, *Parameter control in evolutionary algorithms*, IEEE Transactions On Evolutionary Computation, Vol. 3, No. 2, pp. 124-141, 1999.

[11] A. Aleti, *An Adaptive Approach to Controlling Parameters of Evolutionary Algorithms*, Ph.D. thesis, Swinburne University of Technology, 2012.

[12] K. A. De Jong, *The analysis of the behavior of a class of genetic adaptive systems.* Ph.D. thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[13] J. J. Grefenstette, *Optimization of control parameters for genetic algorithms*, IEEE Trans. Systems, Man, Cybern., Vol. 16, No. 1, pp. 122–128, 1986.

[14] Y. Derlet, *Réglage de la taille de la population de l'algorithme génétique au sein de l'optimisation assistée par modèles de substitution présent dans le logiciel Minamo*, mémoire, Namur, UNamur, 2019.

[15] Cenaero, *Simulation technologies for Aeronautics*, `http://www.cenaero.be/`, consulté le 19 mai 2020.

[16] Cenaero, *Supercalculateur zenobe*, `https://tier1.cenaero.be/fr/zenobe`, consulté le 30 avril 2020.

[17] J. Blanchard, *Agorithmes Évolutionnaires Distribués/Coopératifs*, mémoire, Namur, UNamur, 2015.

[18] Cenaero, *Minamo 3.1.4 Theoretical Manual*, 2020.

[19] A. Auger, *Benchmarking the (1+1) Evolution Strategy with One-Fifth Success Rule on the BBOB-2009 Function Testbed*, ACM-GECCO Genetic and Evolutionary Computation Conference, Canada, 2009.

[20] S. Kern & S.D. Müller & N. Hansen et al., *Learning Probability Distributions in Continuous Evolutionary Algorithms - A Comparative Review*, Natural Computing, Vol. 3, pp. 77-112, 2004.

[21] N. Hansen, D. Arnold & A. Auger, *Evolution strategies*, In : Kacprzyk J., Pedrycz W. (eds.), *Springer Handbook of Computational Intelligence*, pp. 871–898, Springer, Berlin, Heidelberg, 2015.

[22] M.A. Ardeh, *BenchmarkFcns*, `http://benchmarkfcns.xyz/fcns`, consulté le 11 avril 2021.

[23] Regis, Rommel G., *Evolutionnary Programming for High-Dimensional Constrained Expensive Black-Box Optimization Using Radial Basis Functions*, IEEE Transactions on Evolutionary Computation, Vol. 18, No. 3, pp. 326-347, 2014.

[24] Regis, Rommel G., *Constrained Optimization by Radial Basis Function Interpolation for High-Dimensional Expensive Black-Box Problems with Infeasible Initial Points*, Engineering Optimization, Vol. 46, No. 2, pp. 218-243, 2014.

[25] E.D. Dolan & J.J. Moré, *Benchmarking Optimization Software with Performance Profiles*, Mathematical Programming, Vol. 91, No. 2, pp. 201-213, 2002.

[26] W. H. Kruskal & W. W. Wallis, *Use of Ranks in One-Criterion Variance Analysis*, Journal of the American Statistical Association, Vol. 47, No. 260, pp. 583-621, 1952.

[27] O.J. Dunn, *Multiple comparisons using rank sums*, Technometrics, Vol. 6, No. 3, pp. 241-252, 1964.

Annexes

Annexe A - Chapitre 6

Cas-test Ackley_10

GA_Both_GO (2_go)	100.0%
GA_Infill_GO (ifl_go)	100.0%
GA_Offspring_Classic (off_cls_vr)	100.0%
GA_Offspring_GO (off_go)	100.0%
GA_Offspring_Mean (off_mn_vr)	100.0%
GA_Offspring_Median (off_med_vr)	100.0%

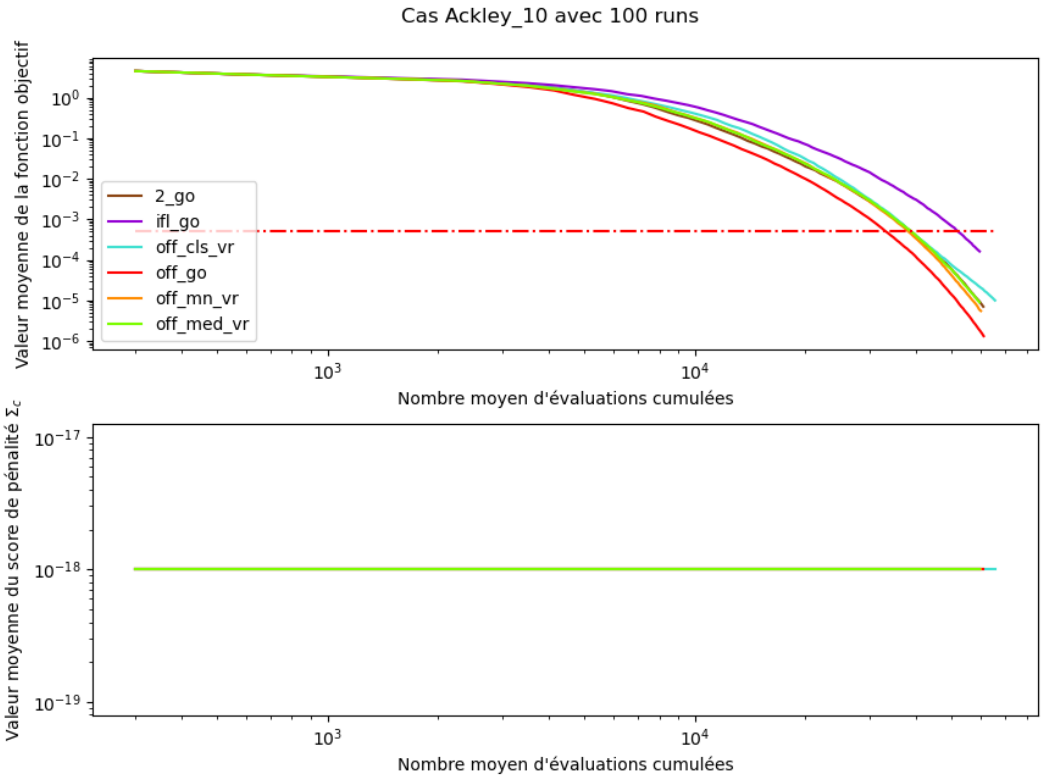
Tableau 1: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Ackley_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_GO (2_go)	[1.49e-06,1.91e-05]	7.01e-06	6.05e-06	0.00e+00
GA_Infill_GO (ifl_go)	[5.99e-05,4.27e-04]	1.61e-04	1.29e-04	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[1.36e-06,4.25e-05]	9.97e-06	8.36e-06	0.00e+00
GA_Offspring_GO (off_go)	[8.18e-08,4.55e-06]	1.32e-06	1.15e-06	0.00e+00
GA_Offspring_Mean (off_mn_vr)	[1.12e-06,1.66e-05]	5.49e-06	4.60e-06	0.00e+00
GA_Offspring_Median (off_med_vr)	[2.40e-06,3.26e-05]	9.01e-06	8.00e-06	0.00e+00

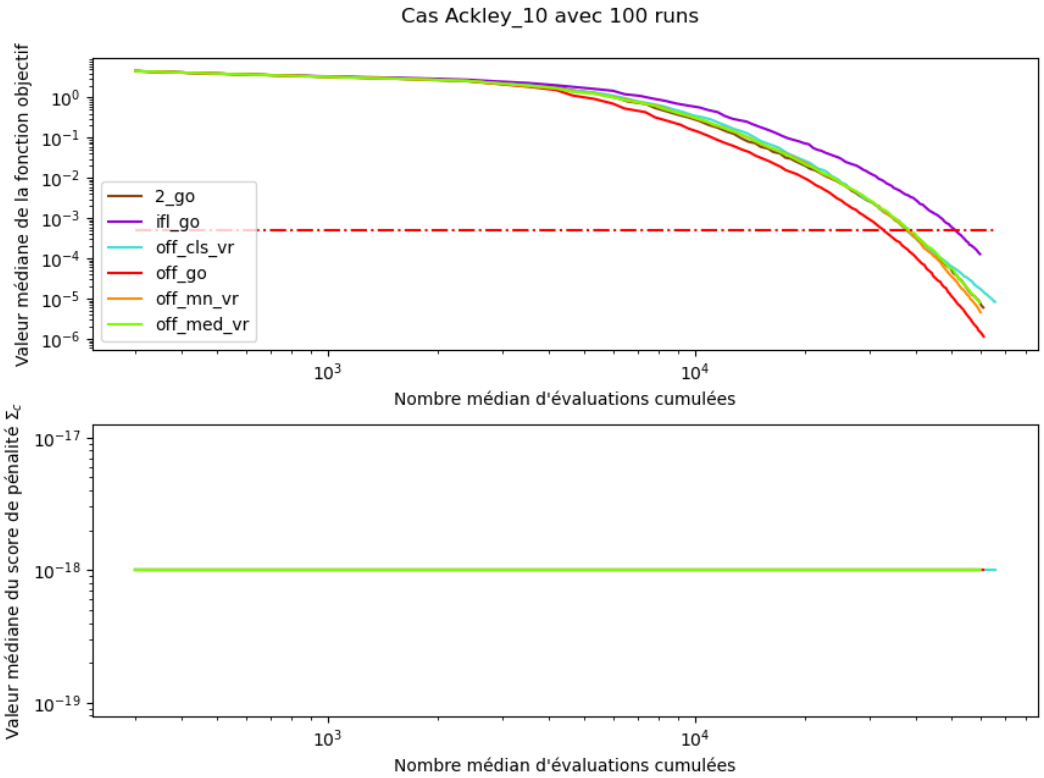
Tableau 2: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Ackley_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	2_go	ifl_go	off_cls_vr	off_go	off_mn_vr	off_med_vr
2_go	1.00e+00	2.69e-26	1.87e-01	3.07e-19	4.84e-01	6.59e-01
ifl_go	2.69e-26	1.00e+00	9.34e-16	7.44e-89	1.81e-37	1.37e-17
off_cls_vr	1.87e-01	9.34e-16	1.00e+00	9.55e-31	5.23e-05	1.00e+00
off_go	3.07e-19	7.44e-89	9.55e-31	1.00e+00	1.63e-11	2.63e-28
off_mn_vr	4.84e-01	1.81e-37	5.23e-05	1.63e-11	1.00e+00	4.86e-04
off_med_vr	6.59e-01	1.37e-17	1.00e+00	2.63e-28	4.86e-04	1.00e+00

Tableau 3: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Ackley_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

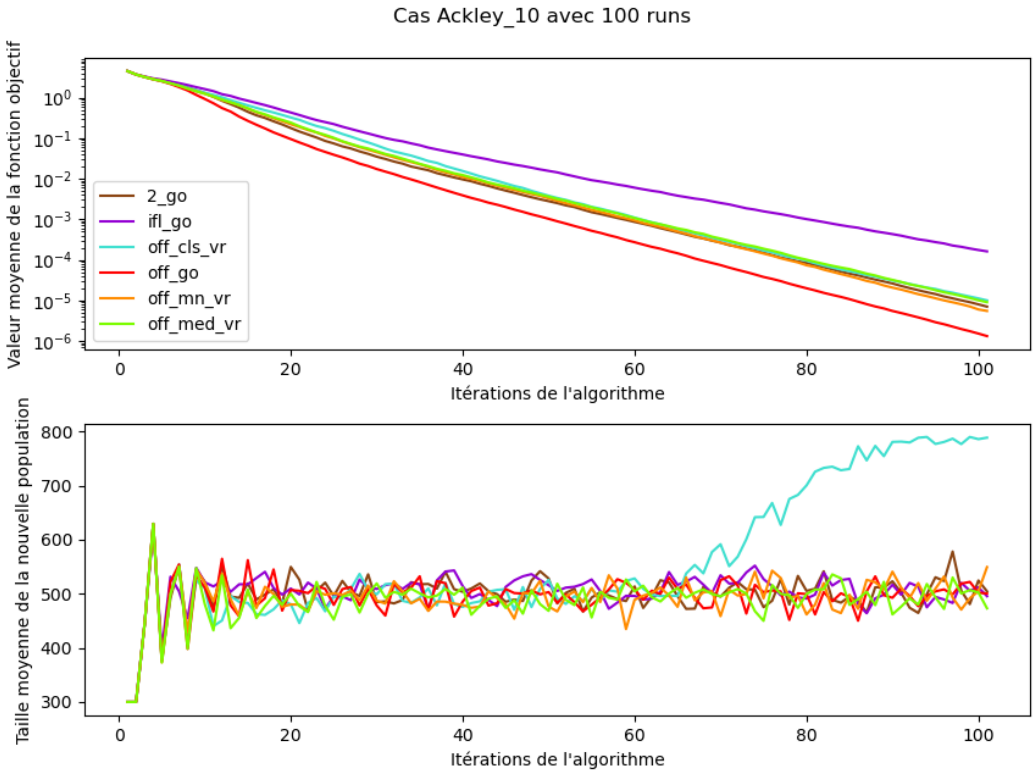


(a) Graphiques des résultats moyens.

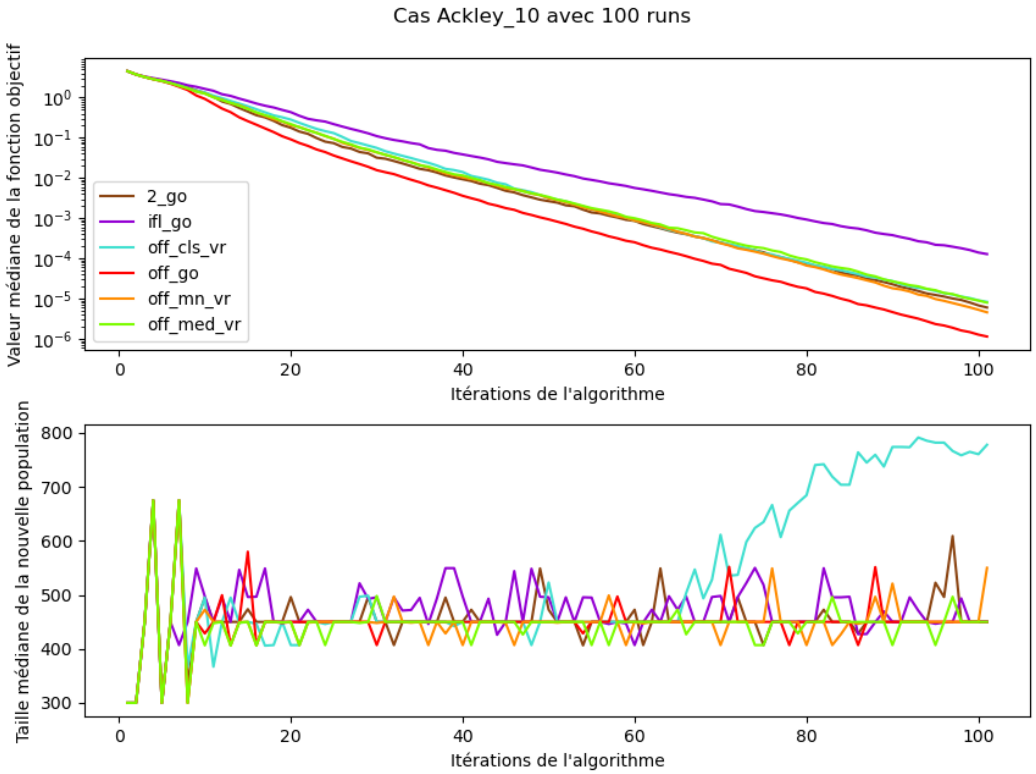


(b) Graphiques des résultats médians.

Figure 1: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Ackley_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 2: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Ackley_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test G2_plog_10

GA_Both_GO (2_go)	86.0%
GA_Infill_GO (ifl_go)	86.0%
GA_Offspring_Classic (off_cls_vr)	75.0%
GA_Offspring_GO (off_go)	69.0%
GA_Offspring_Mean (off_mn_vr)	82.0%
GA_Offspring_Median (off_med_vr)	75.0%

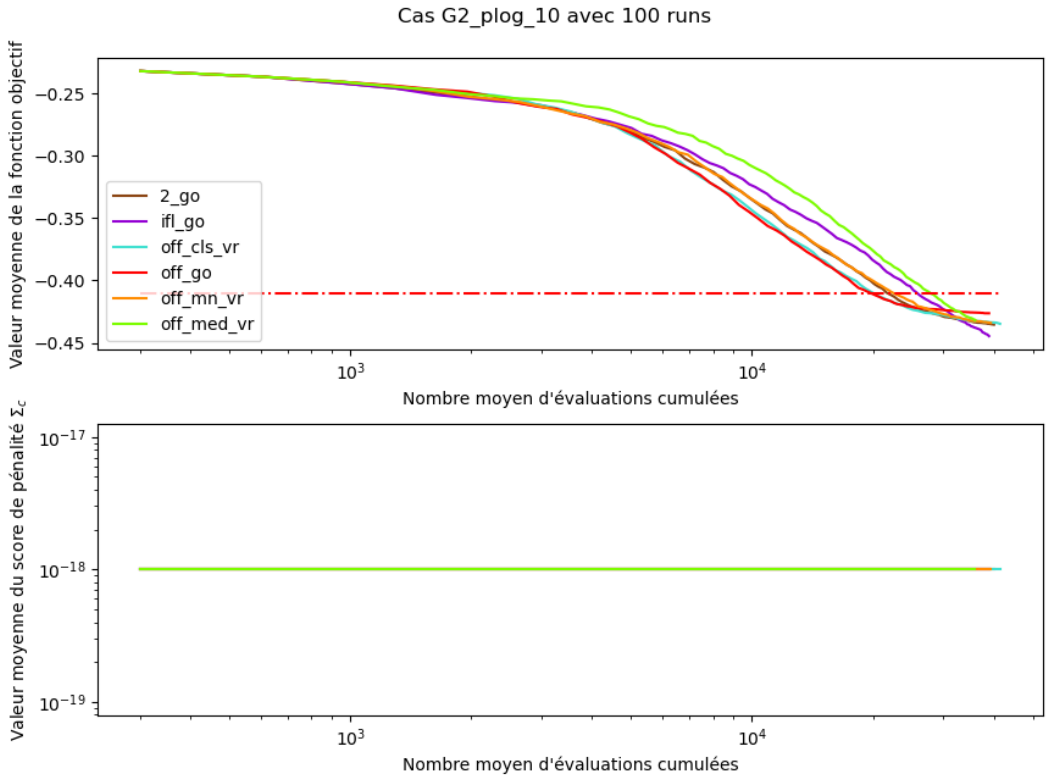
Tableau 4: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test G2_plog_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_GO (2_go)	[2.72e-01,4.79e-01]	4.12e-01	4.18e-01	0.00e+00
GA_Infill_GO (ifl_go)	[2.11e-01,4.79e-01]	4.00e-01	4.04e-01	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[8.04e-02,4.80e-01]	4.13e-01	4.23e-01	0.00e+00
GA_Offspring_GO (off_go)	[3.43e-01,5.04e-01]	4.24e-01	4.29e-01	0.00e+00
GA_Offspring_Mean (off_mn_vr)	[2.74e-01,4.96e-01]	4.14e-01	4.21e-01	0.00e+00
GA_Offspring_Median (off_med_vr)	[2.95e-01,4.96e-01]	4.17e-01	4.22e-01	0.00e+00

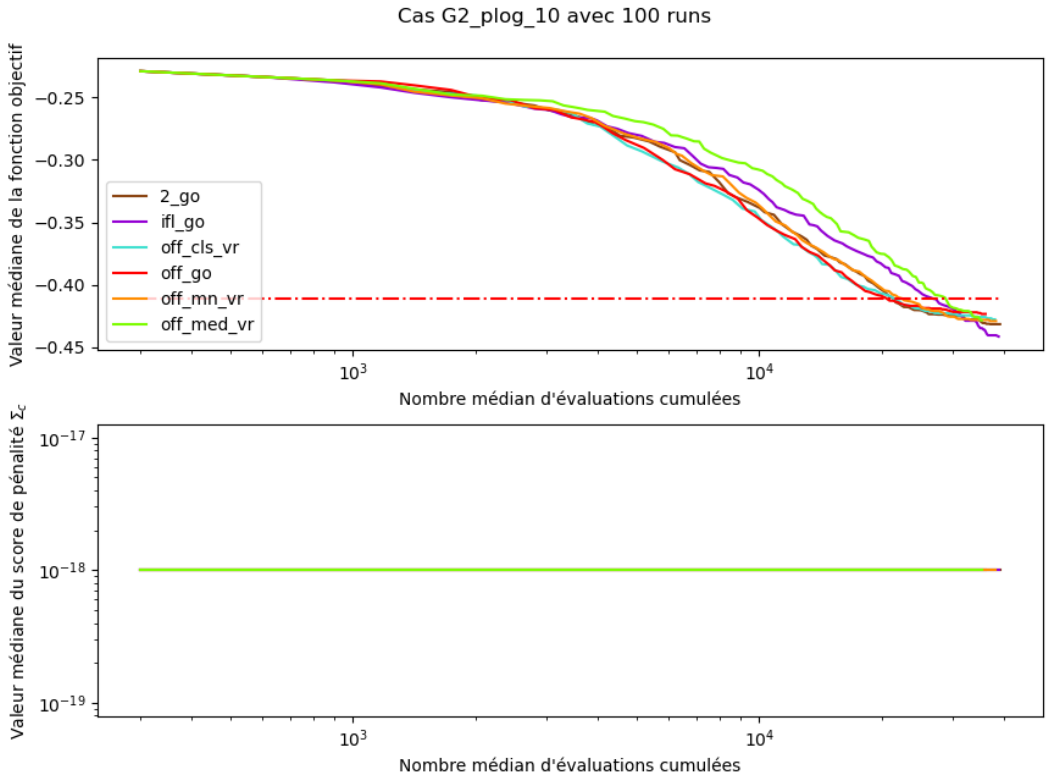
Tableau 5: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test G2_plog_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	2_go	ifl_go	off_cls_vr	off_go	off_mn_vr	off_med_vr
2_go	1.00e+00	1.00e+00	1.00e+00	3.32e-01	1.00e+00	1.00e+00
ifl_go	1.00e+00	1.00e+00	7.22e-02	7.51e-04	2.94e-01	5.33e-02
off_cls_vr	1.00e+00	7.22e-02	1.00e+00	1.00e+00	1.00e+00	1.00e+00
off_go	3.32e-01	7.51e-04	1.00e+00	1.00e+00	1.00e+00	1.00e+00
off_mn_vr	1.00e+00	2.94e-01	1.00e+00	1.00e+00	1.00e+00	1.00e+00
off_med_vr	1.00e+00	5.33e-02	1.00e+00	1.00e+00	1.00e+00	1.00e+00

Tableau 6: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test G2_plog_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

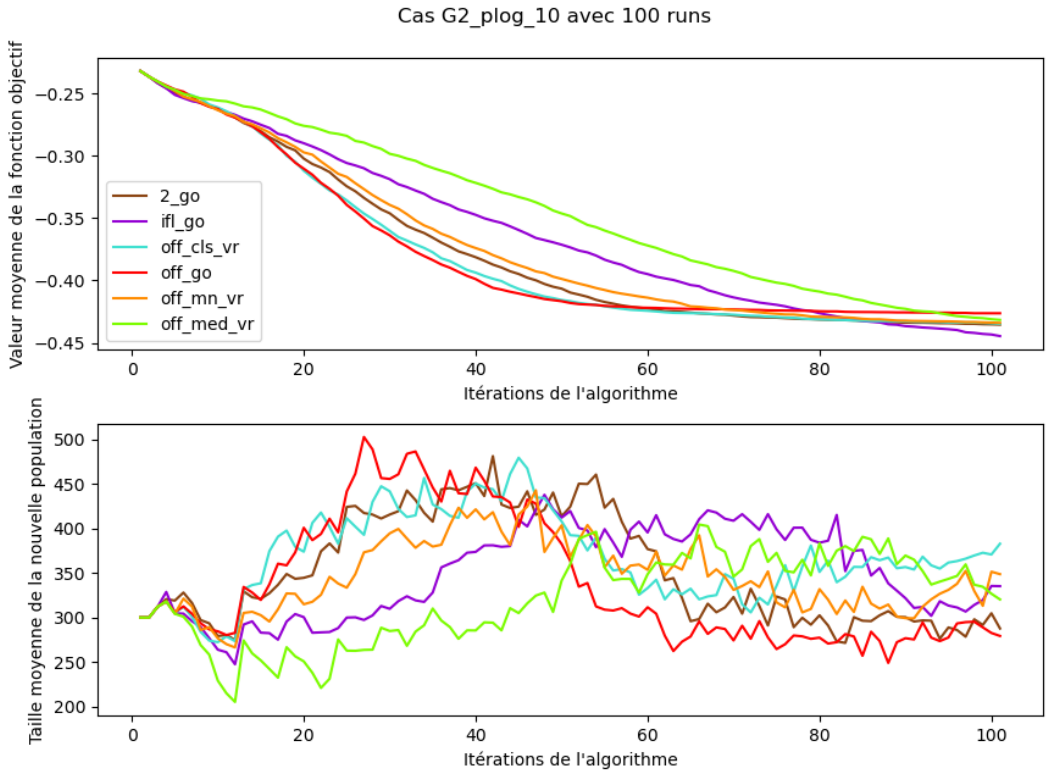


(a) Graphiques des résultats moyens.

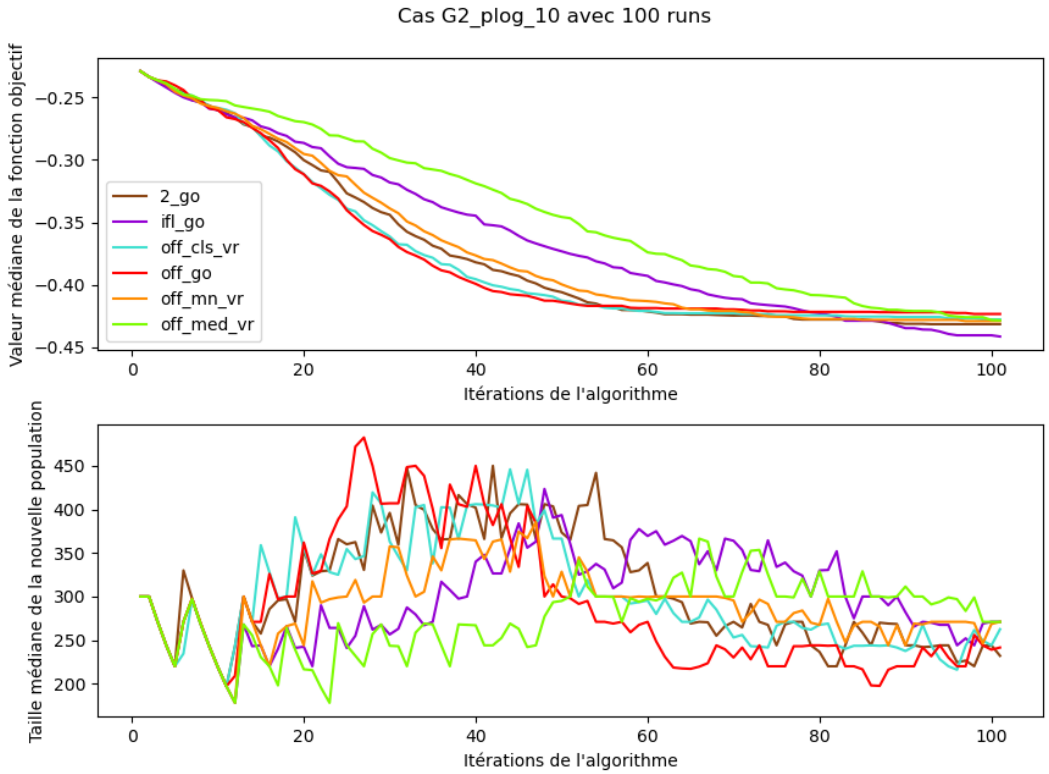


(b) Graphiques des résultats médians.

FIGURE 3: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test G2_plog_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 4: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test G2_plog_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test G7

GA_Both_GO (2_go)	100.0%
GA_Infill_GO (ifl_go)	99.0%
GA_Offspring_Classic (off_cls_vr)	76.0%
GA_Offspring_GO (off_go)	100.0%
GA_Offspring_Mean (off_mn_vr)	100.0%
GA_Offspring_Median (off_med_vr)	75.0%

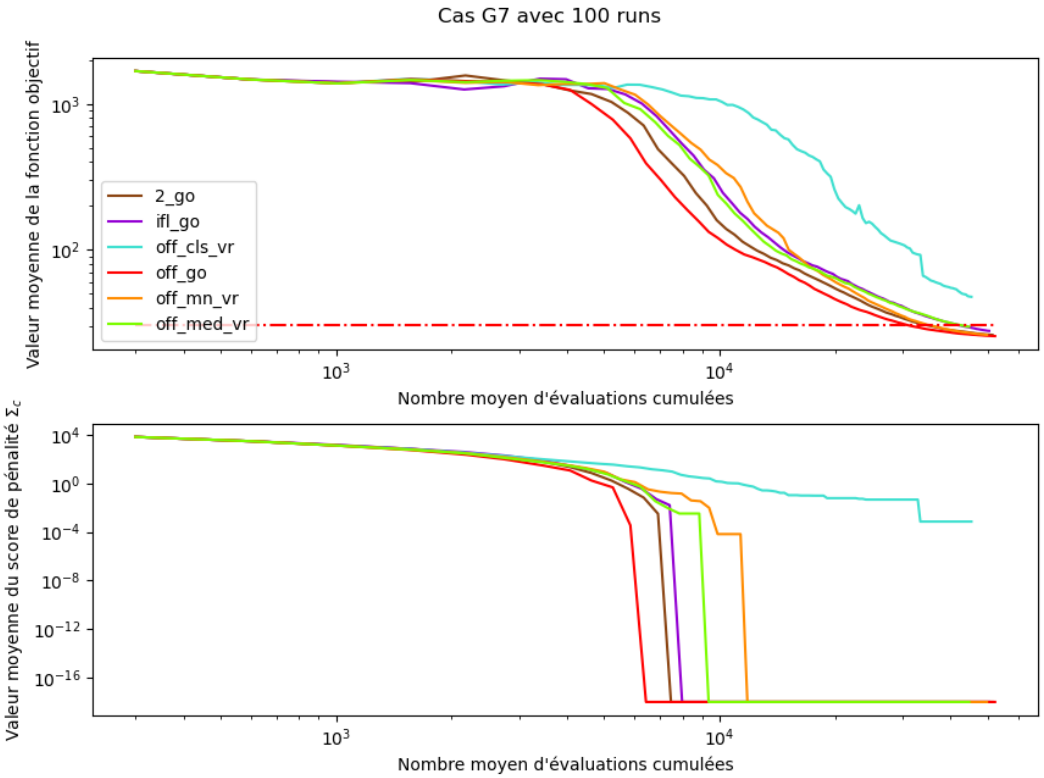
Tableau 7: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test G7.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_GO (2_go)	[3.20e-02,1.38e-01]	6.77e-02	6.62e-02	0.00e+00
GA_Infill_GO (ifl_go)	[5.62e-02,2.65e-01]	1.38e-01	1.32e-01	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[4.87e-02,5.73e+01]	9.50e-01	1.65e-01	0.00e+00
GA_Offspring_GO (off_go)	[2.16e-02,1.65e-01]	4.79e-02	4.52e-02	0.00e+00
GA_Offspring_Mean (off_mn_vr)	[3.26e-02,2.08e-01]	6.92e-02	6.17e-02	0.00e+00
GA_Offspring_Median (off_med_vr)	[1.05e-01,4.03e-01]	2.15e-01	2.08e-01	0.00e+00

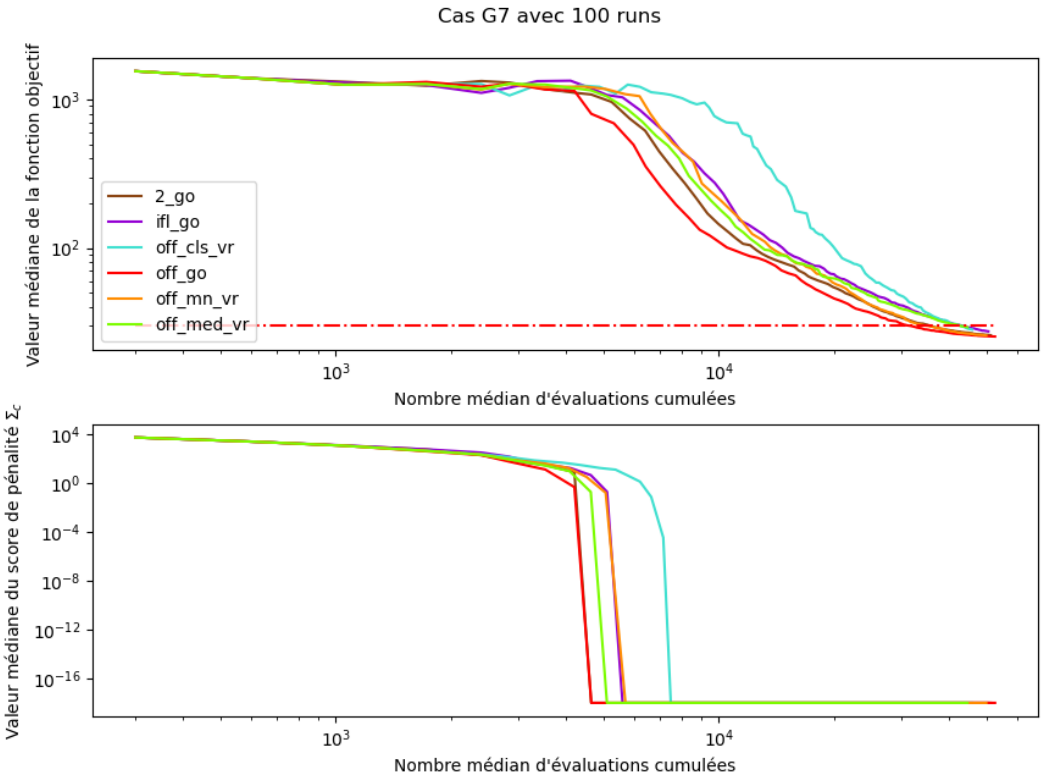
Tableau 8: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test G7. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	2_go	ifl_go	off_cls_vr	off_go	off_mn_vr	off_med_vr
2_go	1.00e+00	2.03e-14	2.08e-22	1.50e-03	1.00e+00	1.29e-34
ifl_go	2.03e-14	1.00e+00	6.51e-01	2.27e-31	5.02e-15	1.02e-04
off_cls_vr	2.08e-22	6.51e-01	1.00e+00	9.62e-43	3.66e-23	1.97e-01
off_go	1.50e-03	2.27e-31	9.62e-43	1.00e+00	3.00e-03	4.07e-59
off_mn_vr	1.00e+00	5.02e-15	3.66e-23	3.00e-03	1.00e+00	1.49e-35
off_med_vr	1.29e-34	1.02e-04	1.97e-01	4.07e-59	1.49e-35	1.00e+00

Tableau 9: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test G7. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

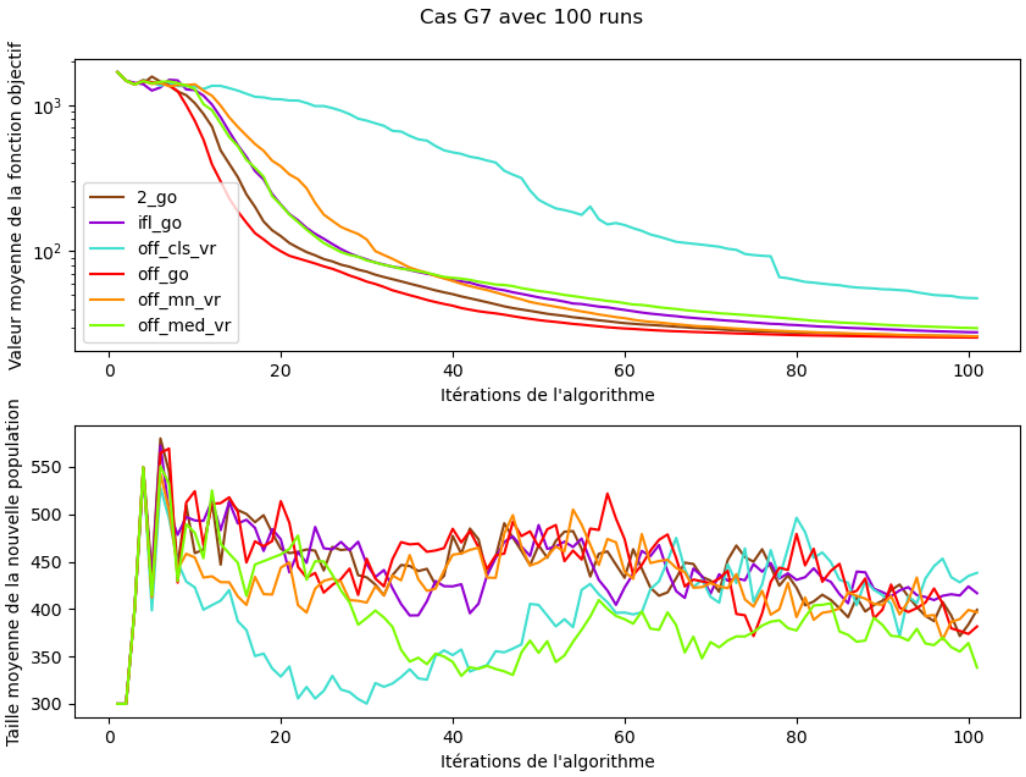


(a) Graphiques des résultats moyens.

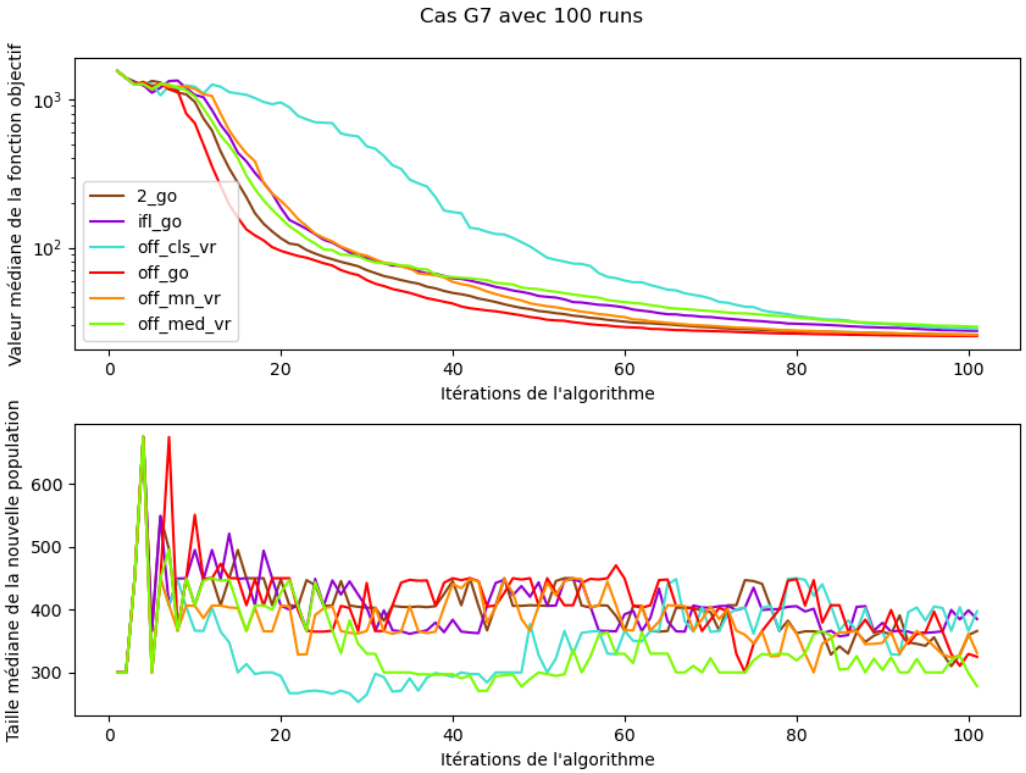


(b) Graphiques des résultats médians.

Figure 5: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test G7. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 6: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test G7. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test G10

GA_Both_GO (2_go)	97.0%
GA_Infill_GO (ifl_go)	48.0%
GA_Offspring_Classic (off_cls_vr)	72.0%
GA_Offspring_GO (off_go)	99.0%
GA_Offspring_Mean (off_mn_vr)	93.0%
GA_Offspring_Median (off_med_vr)	78.0%

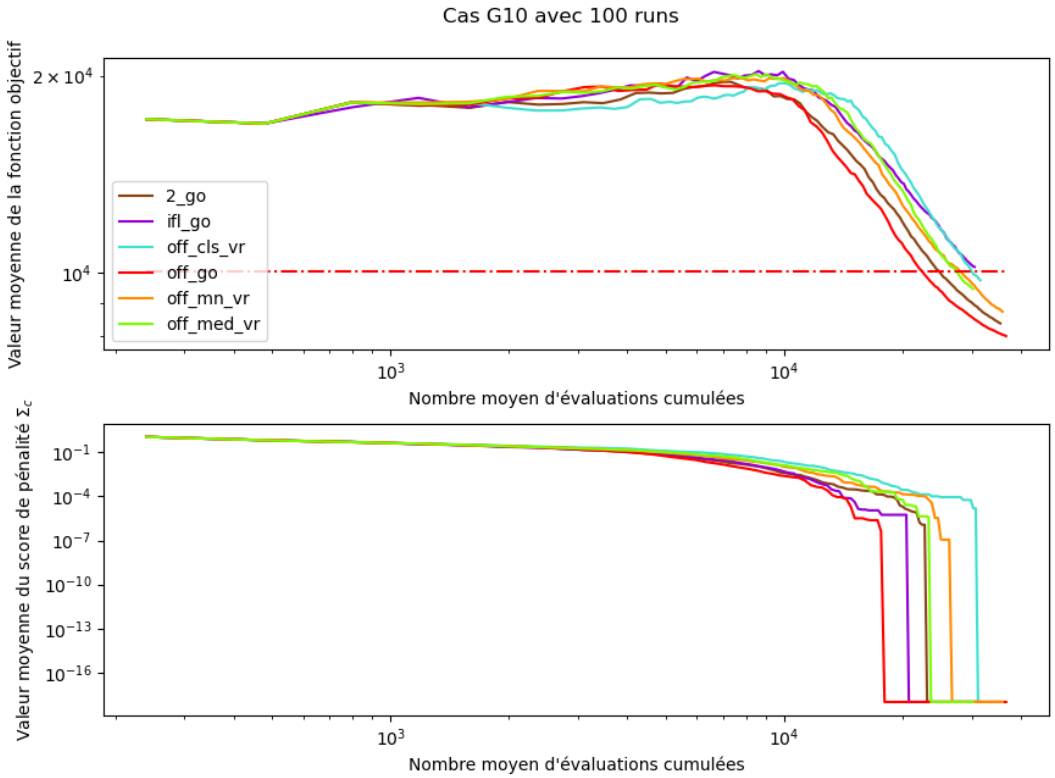
Tableau 10: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test G10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_GO (2_go)	[8.13e-02,1.03e+00]	1.88e-01	1.51e-01	0.00e+00
GA_Infill_GO (ifl_go)	[1.49e-01,1.16e+00]	4.49e-01	4.32e-01	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[1.27e-01,1.54e+00]	3.82e-01	3.20e-01	0.00e+00
GA_Offspring_GO (off_go)	[4.96e-02,4.91e-01]	1.36e-01	1.14e-01	0.00e+00
GA_Offspring_Mean (off_mn_vr)	[8.20e-02,1.22e+00]	2.38e-01	1.93e-01	0.00e+00
GA_Offspring_Median (off_med_vr)	[1.30e-01,7.03e-01]	3.41e-01	3.30e-01	0.00e+00

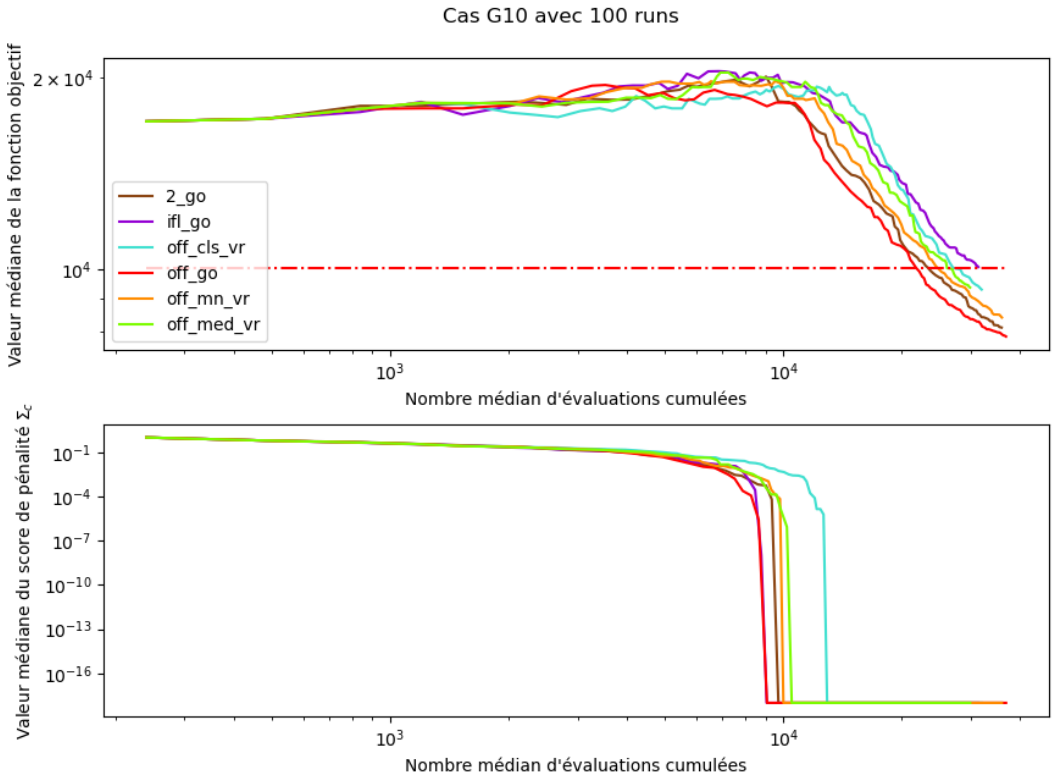
Tableau 11: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test G10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	2_go	ifl_go	off_cls_vr	off_go	off_mn_vr	off_med_vr
2_go	1.00e+00	2.59e-27	2.00e-15	6.27e-02	9.88e-02	1.46e-16
ifl_go	2.59e-27	1.00e+00	7.64e-02	5.76e-43	9.87e-16	1.89e-01
off_cls_vr	2.00e-15	7.64e-02	1.00e+00	1.28e-27	4.21e-07	1.00e+00
off_go	6.27e-02	5.76e-43	1.28e-27	1.00e+00	3.58e-07	3.91e-29
off_mn_vr	9.88e-02	9.87e-16	4.21e-07	3.58e-07	1.00e+00	6.96e-08
off_med_vr	1.46e-16	1.89e-01	1.00e+00	3.91e-29	6.96e-08	1.00e+00

Tableau 12: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test G10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

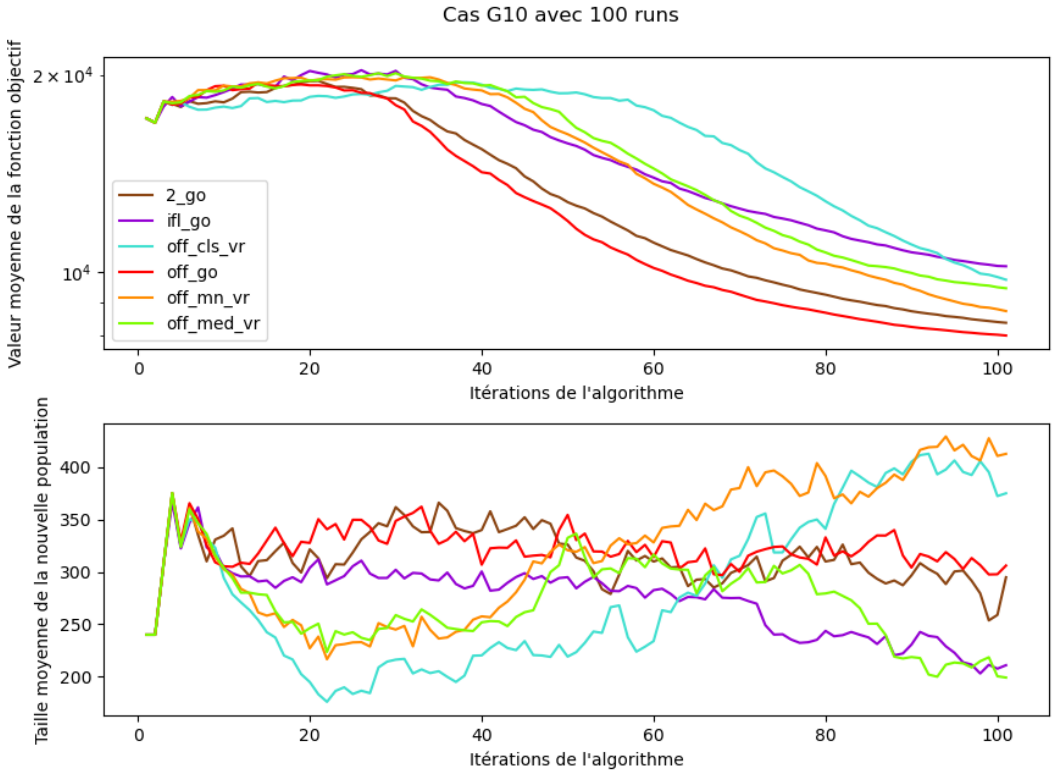


(a) Graphiques des résultats moyens.

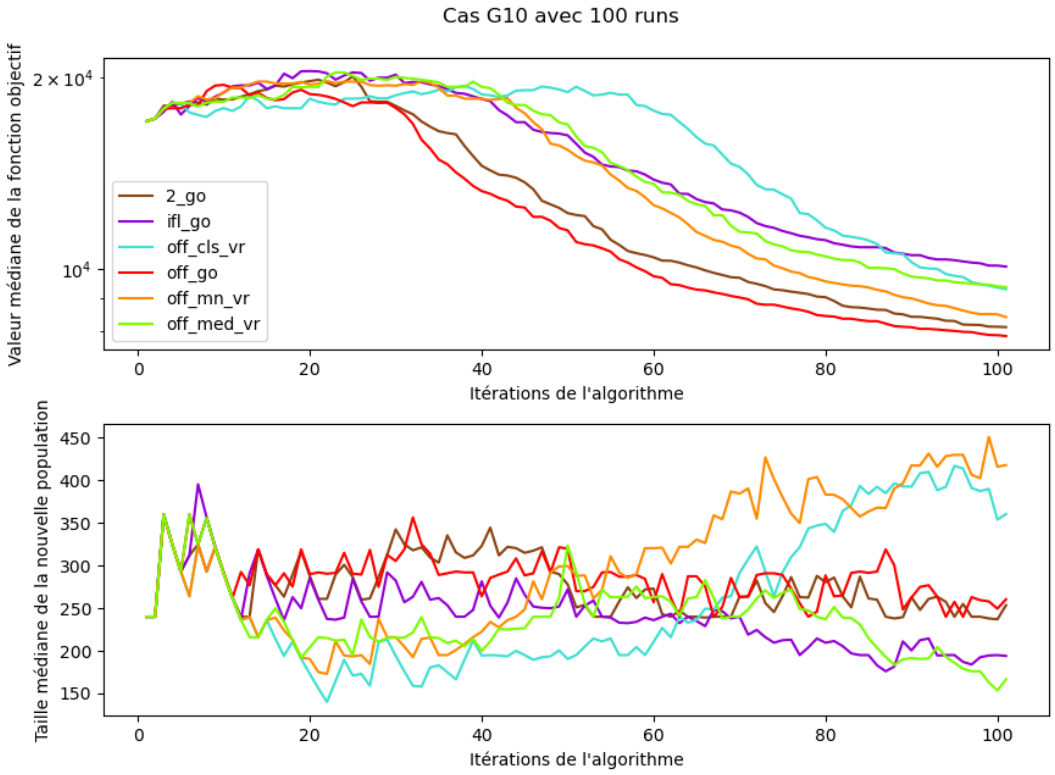


(b) Graphiques des résultats médians.

FIGURE 7: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test G10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 8: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test G10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test Rastrigin_10

GA_Both_GO (2_go)	100.0%
GA_Infill_GO (ifl_go)	95.0%
GA_Offspring_Classic (off_cls_vr)	77.0%
GA_Offspring_GO (off_go)	100.0%
GA_Offspring_Mean (off_mn_vr)	89.0%
GA_Offspring_Median (off_med_vr)	91.0%

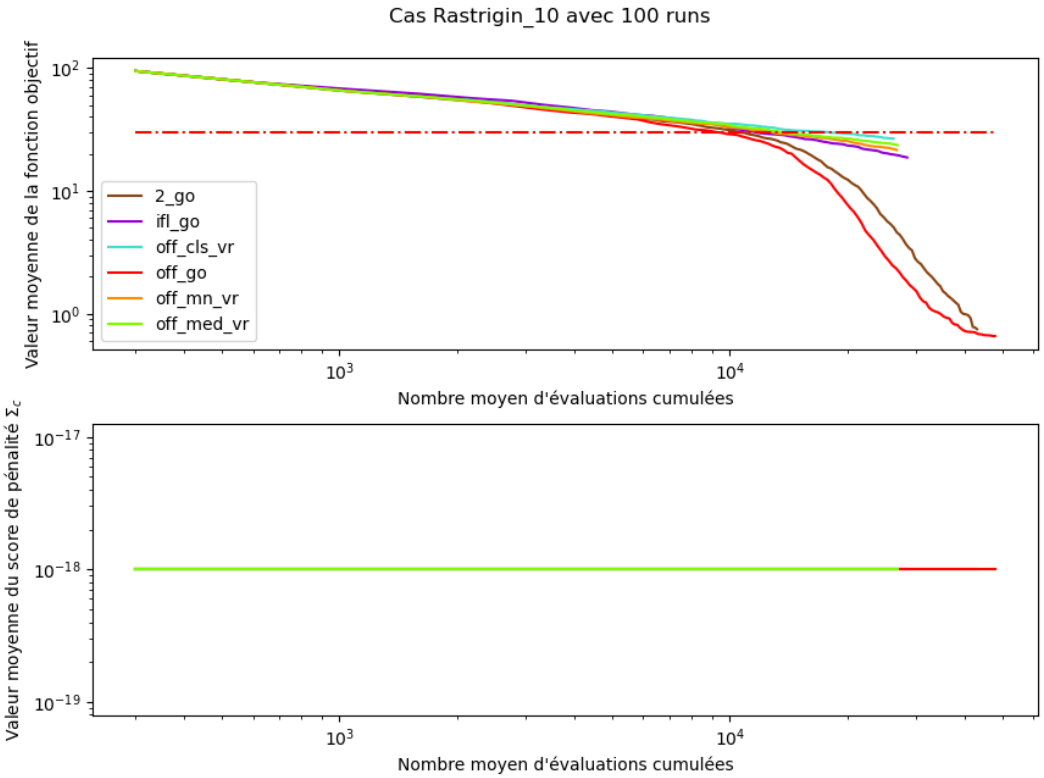
Tableau 13: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Rastrigin_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_GO (2_go)	[6.33e-06,1.25e+01]	7.48e-01	9.88e-02	0.00e+00
GA_Infill_GO (ifl_go)	[2.71e-02,3.56e+01]	1.87e+01	2.14e+01	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[7.04e+00,4.07e+01]	2.67e+01	2.72e+01	0.00e+00
GA_Offspring_GO (off_go)	[3.60e-08,5.98e+00]	6.57e-01	9.65e-05	0.00e+00
GA_Offspring_Mean (off_mn_vr)	[4.80e+00,3.58e+01]	2.16e+01	2.18e+01	0.00e+00
GA_Offspring_Median (off_med_vr)	[1.17e+01,3.55e+01]	2.36e+01	2.37e+01	0.00e+00

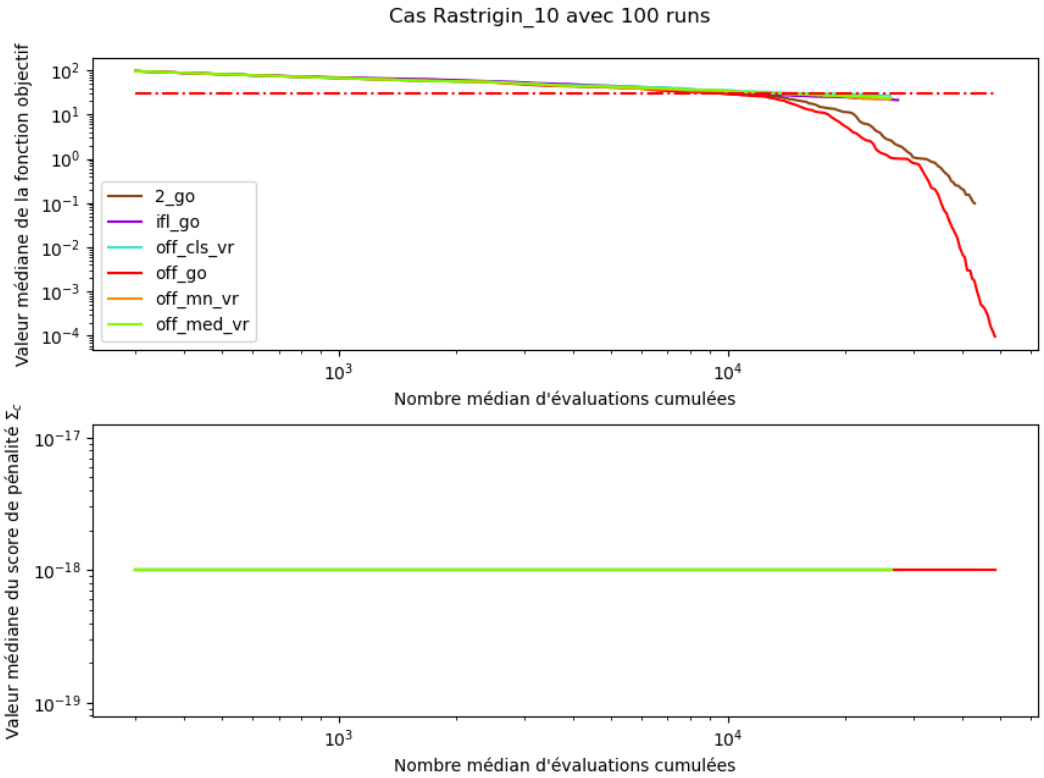
Tableau 14: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Rastrigin_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	2_go	ifl_go	off_cls_vr	off_go	off_mn_vr	off_med_vr
2_go	1.00e+00	3.07e-19	1.75e-44	1.00e+00	1.08e-23	7.95e-31
ifl_go	3.07e-19	1.00e+00	1.28e-05	2.49e-25	1.00e+00	1.79e-01
off_cls_vr	1.75e-44	1.28e-05	1.00e+00	1.55e-53	1.53e-03	2.40e-01
off_go	1.00e+00	2.49e-25	1.55e-53	1.00e+00	2.09e-30	1.99e-38
off_mn_vr	1.08e-23	1.00e+00	1.53e-03	2.09e-30	1.00e+00	1.00e+00
off_med_vr	7.95e-31	1.79e-01	2.40e-01	1.99e-38	1.00e+00	1.00e+00

Tableau 15: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Rastrigin_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

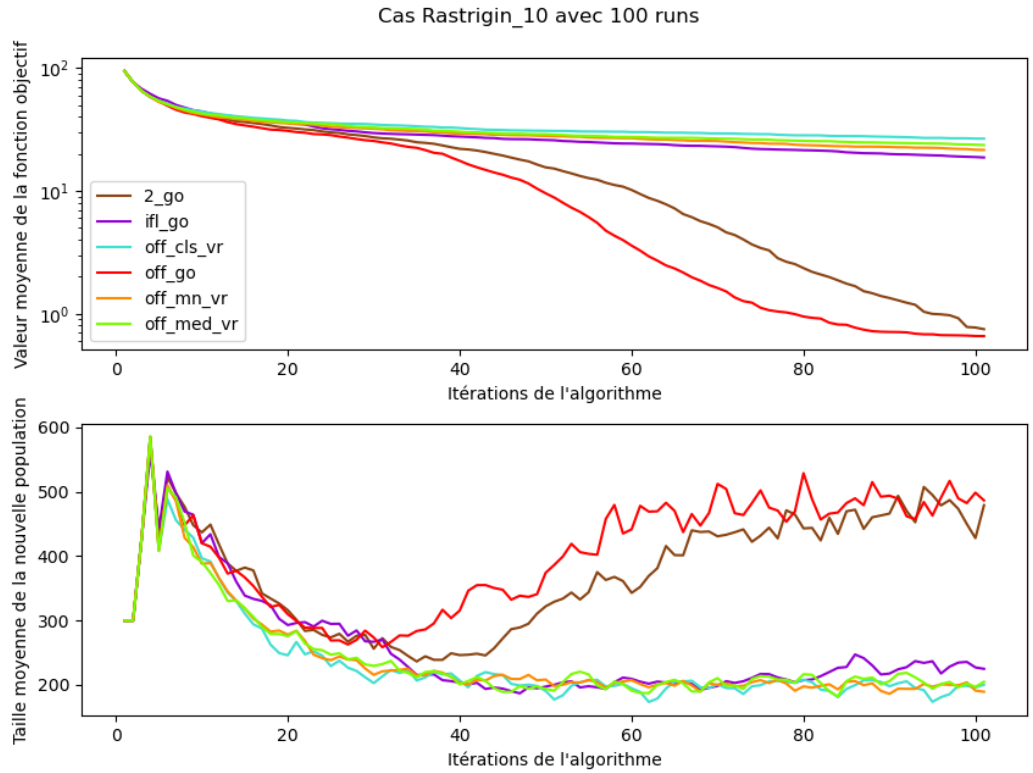


(a) Graphiques des résultats moyens.

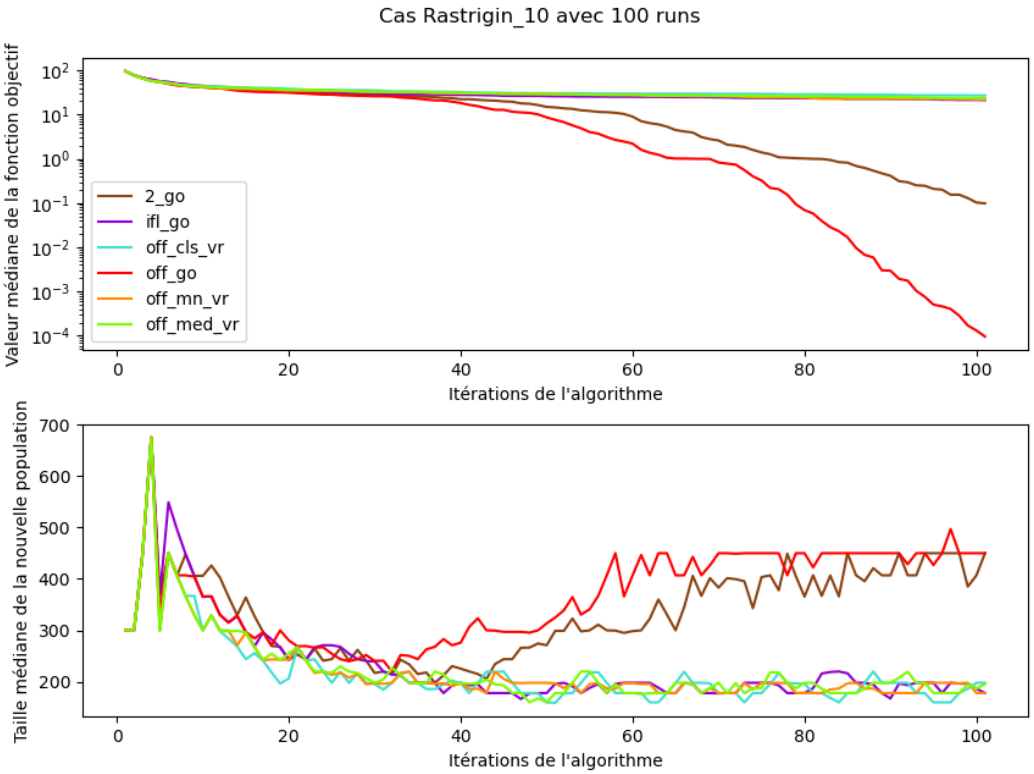


(b) Graphiques des résultats médians.

FIGURE 9: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Rastrigin_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 10: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Rastrigin_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test Rosenbrock_10

GA_Both_GO (2_go)	94.0%
GA_Infill_GO (ifl_go)	57.0%
GA_Offspring_Classic (off_cls_vr)	83.0%
GA_Offspring_GO (off_go)	91.0%
GA_Offspring_Mean (off_mn_vr)	96.0%
GA_Offspring_Median (off_med_vr)	98.0%

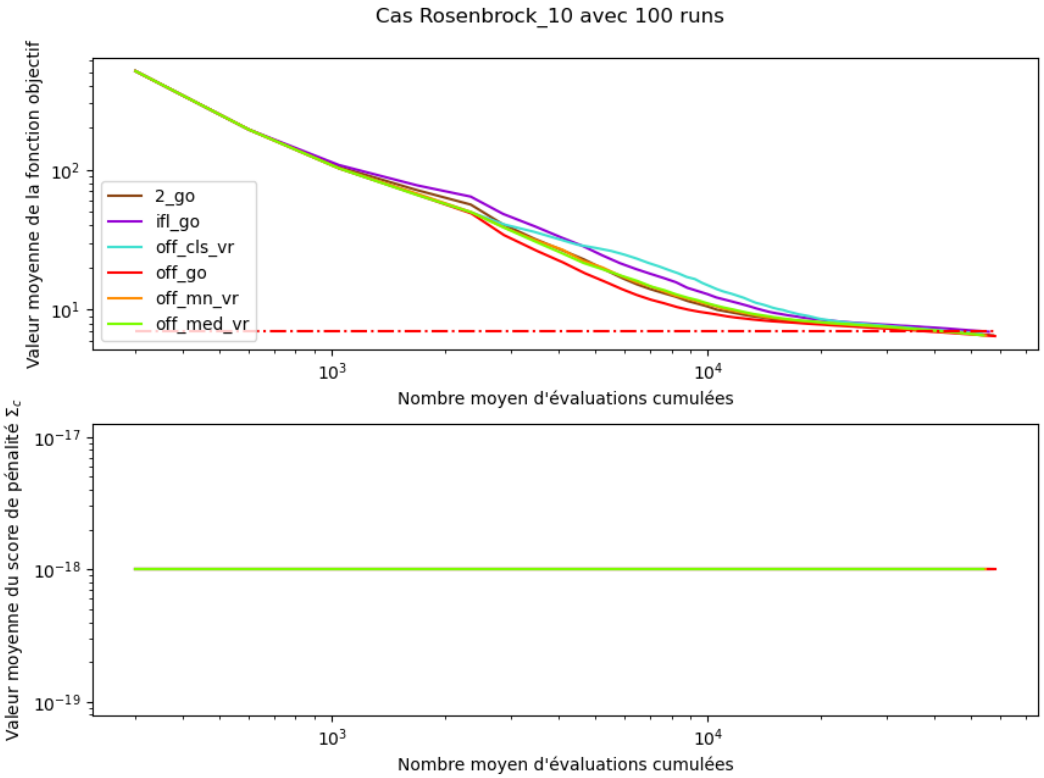
Tableau 16: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Rosenbrock_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_GO (2_go)	[5.69e+00,7.14e+00]	6.56e+00	6.58e+00	0.00e+00
GA_Infill_GO (ifl_go)	[6.16e+00,7.45e+00]	6.97e+00	6.98e+00	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[6.06e+00,8.49e+00]	6.72e+00	6.72e+00	0.00e+00
GA_Offspring_GO (off_go)	[5.26e+00,7.24e+00]	6.49e+00	6.52e+00	0.00e+00
GA_Offspring_Mean (off_mn_vr)	[5.82e+00,9.20e+00]	6.55e+00	6.55e+00	0.00e+00
GA_Offspring_Median (off_med_vr)	[5.94e+00,7.30e+00]	6.57e+00	6.57e+00	0.00e+00

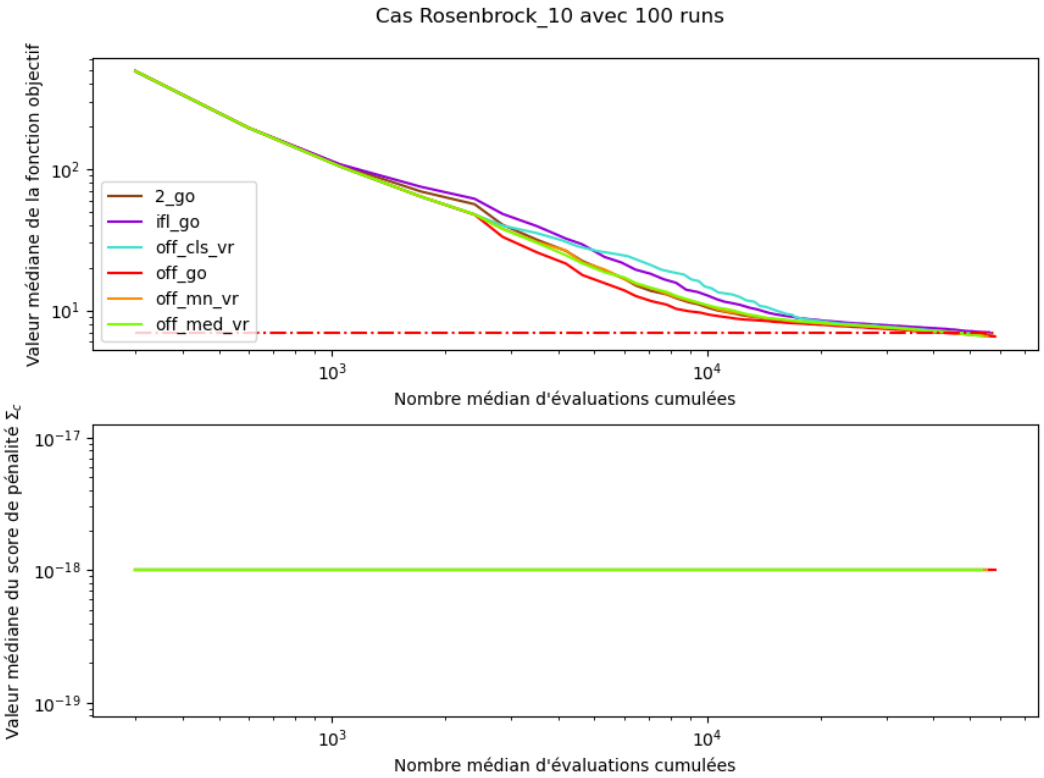
Tableau 17: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Rosenbrock_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	2_go	ifl_go	off_cls_vr	off_go	off_mn_vr	off_med_vr
2_go	1.00e+00	7.75e-17	8.94e-02	1.00e+00	1.00e+00	1.00e+00
ifl_go	7.75e-17	1.00e+00	5.47e-08	2.61e-22	6.27e-21	3.18e-18
off_cls_vr	8.94e-02	5.47e-08	1.00e+00	6.55e-04	2.48e-03	2.83e-02
off_go	1.00e+00	2.61e-22	6.55e-04	1.00e+00	1.00e+00	1.00e+00
off_mn_vr	1.00e+00	6.27e-21	2.48e-03	1.00e+00	1.00e+00	1.00e+00
off_med_vr	1.00e+00	3.18e-18	2.83e-02	1.00e+00	1.00e+00	1.00e+00

Tableau 18: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Rosenbrock_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

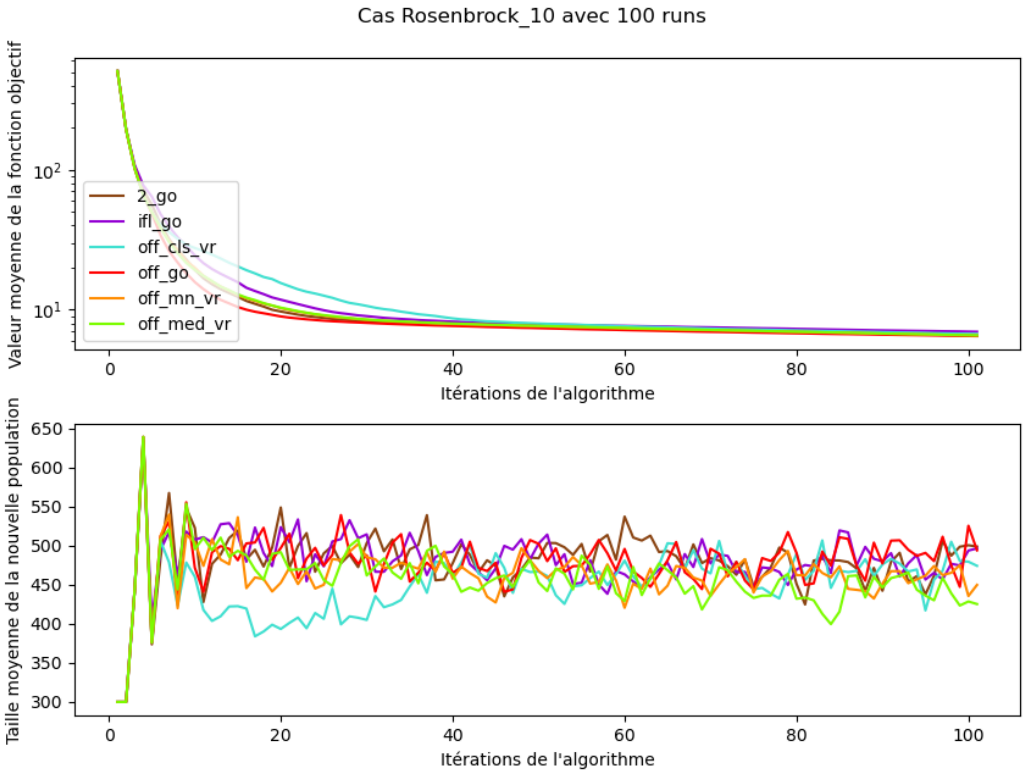


(a) Graphiques des résultats moyens.

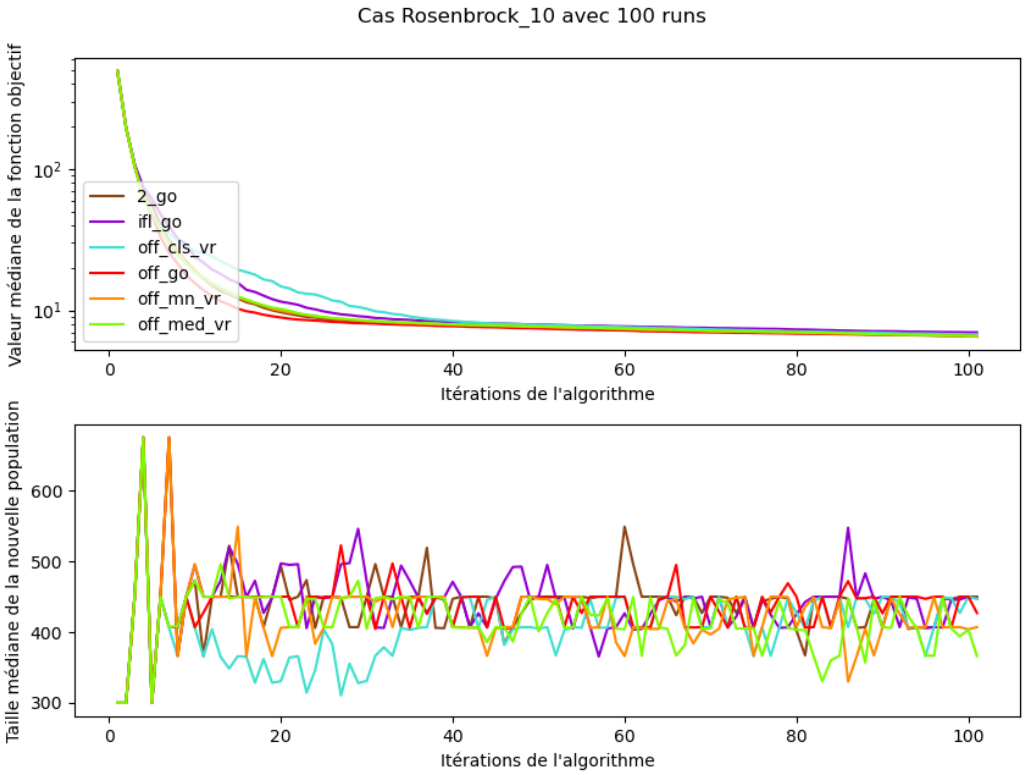


(b) Graphiques des résultats médians.

FIGURE 11: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Rosenbrock_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 12: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Rosenbrock_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test Schwefel_10

GA_Both_GO (2_go)	91.0%
GA_Infill_GO (ifl_go)	95.0%
GA_Offspring_Classic (off_cls_vr)	78.0%
GA_Offspring_GO (off_go)	93.0%
GA_Offspring_Mean (off_mn_vr)	82.0%
GA_Offspring_Median (off_med_vr)	84.0%

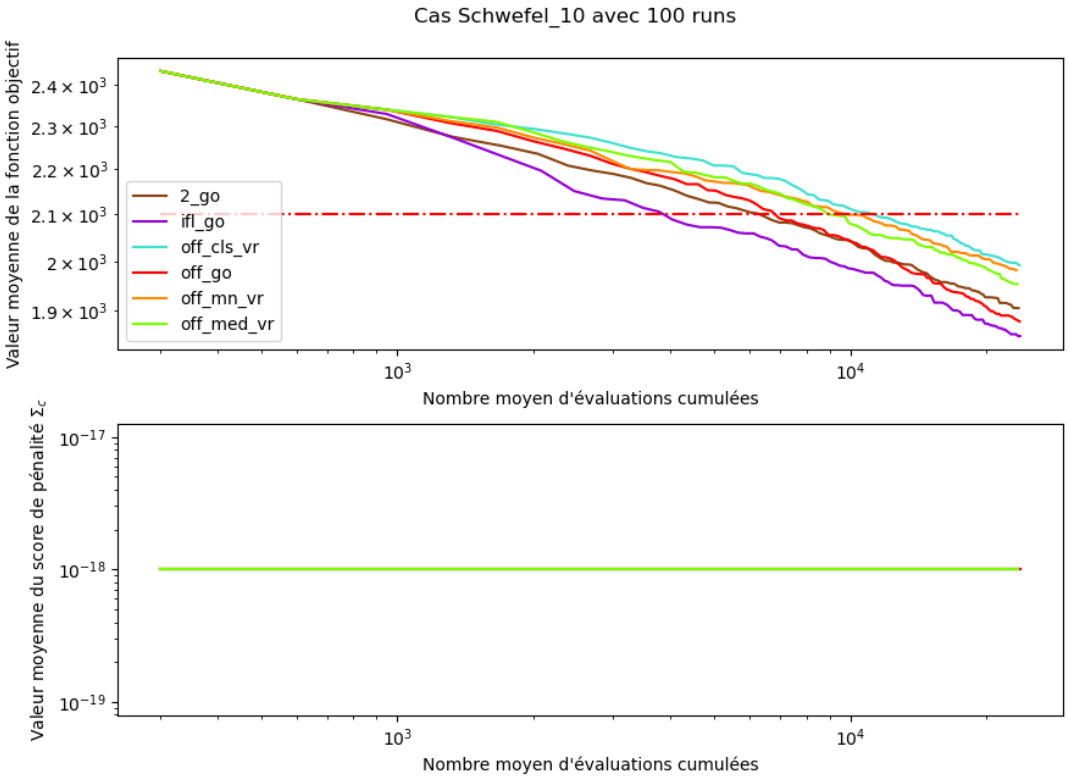
Tableau 19: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Schwefel_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_Both_GO (2_go)	[1.33e+03,2.21e+03]	1.91e+03	1.94e+03	0.00e+00
GA_Infill_GO (ifl_go)	[1.23e+03,2.17e+03]	1.85e+03	1.86e+03	0.00e+00
GA_Offspring_Classic (off_cls_vr)	[1.57e+03,2.27e+03]	1.99e+03	2.00e+03	0.00e+00
GA_Offspring_GO (off_go)	[1.10e+03,2.19e+03]	1.88e+03	1.91e+03	0.00e+00
GA_Offspring_Mean (off_mn_vr)	[1.63e+03,2.27e+03]	1.98e+03	1.99e+03	0.00e+00
GA_Offspring_Median (off_med_vr)	[1.44e+03,2.26e+03]	1.95e+03	1.98e+03	0.00e+00

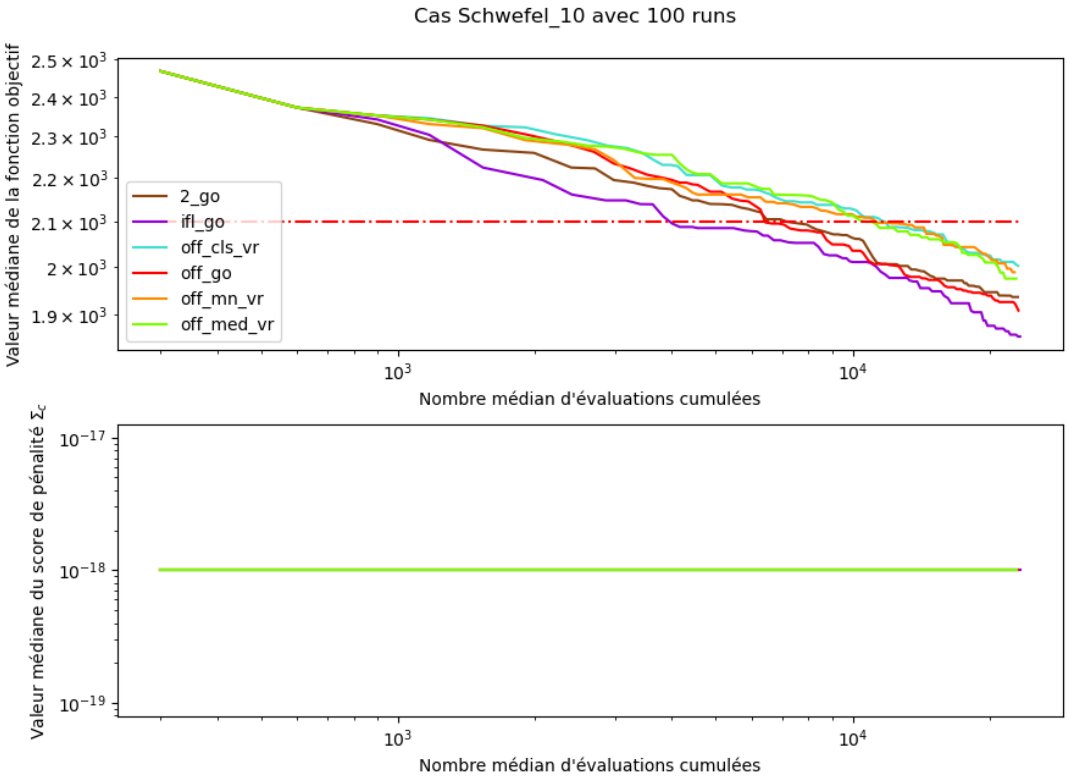
Tableau 20: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Schwefel_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	2_go	ifl_go	off_cls_vr	off_go	off_mn_vr	off_med_vr
2_go	1.00e+00	5.96e-01	1.08e-03	1.00e+00	1.11e-02	3.72e-01
ifl_go	5.96e-01	1.00e+00	2.50e-08	1.00e+00	8.44e-07	2.55e-04
off_cls_vr	1.08e-03	2.50e-08	1.00e+00	1.11e-05	1.00e+00	1.00e+00
off_go	1.00e+00	1.00e+00	1.11e-05	1.00e+00	2.00e-04	1.89e-02
off_mn_vr	1.11e-02	8.44e-07	1.00e+00	2.00e-04	1.00e+00	1.00e+00
off_med_vr	3.72e-01	2.55e-04	1.00e+00	1.89e-02	1.00e+00	1.00e+00

Tableau 21: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Schwefel_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

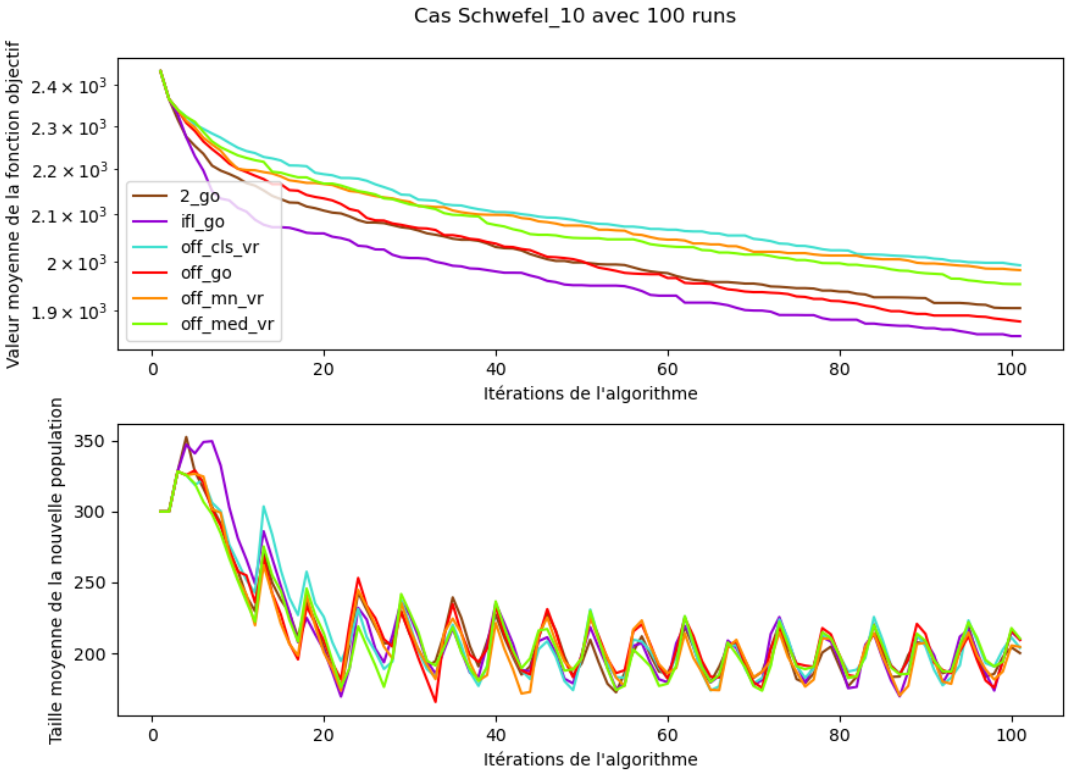


(a) Graphiques des résultats moyens.

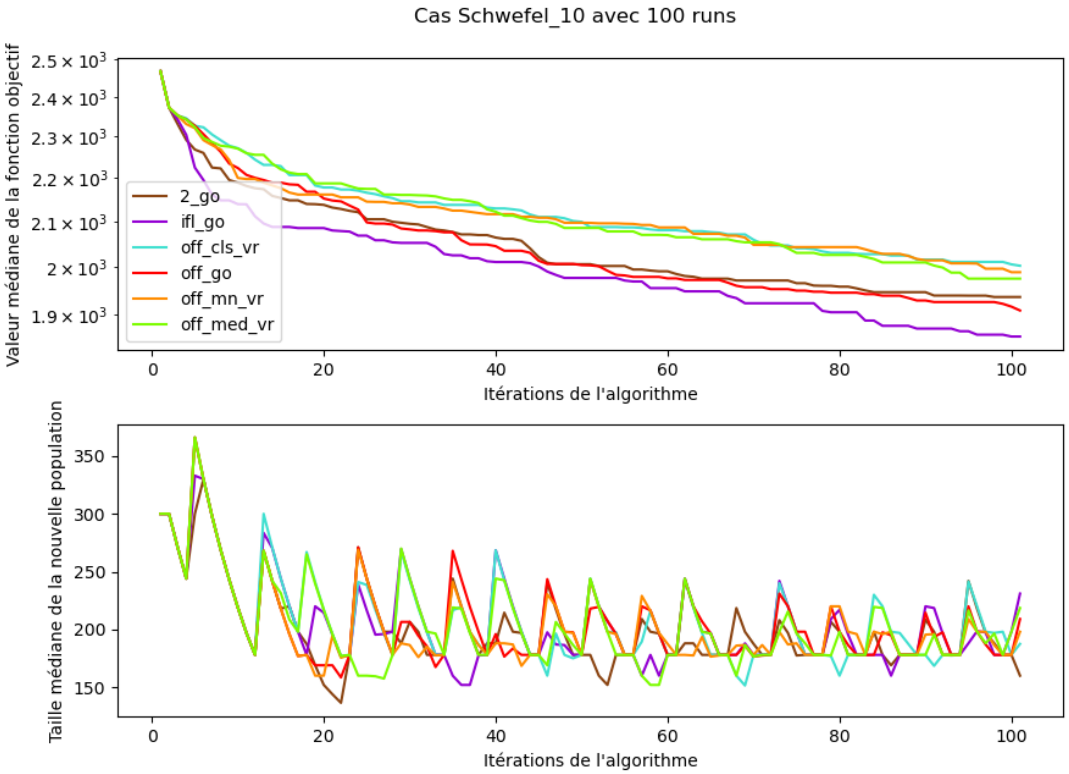


(b) Graphiques des résultats médians.

Figure 13: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Schwefel_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 14: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Schwefel_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Annexe B - Chapitre 7

Cas-test Ackley_10

GA_10 (GA_10)	57.0%
GA_30 (GA_30)	100.0%
GA_50 (GA_50)	100.0%
GA_100 (GA_100)	100.0%
GA_OneFifth (onefifth)	100.0%
GA_Offspring_GO (off_go)	100.0%
GA_Both_GO (2_go)	100.0%

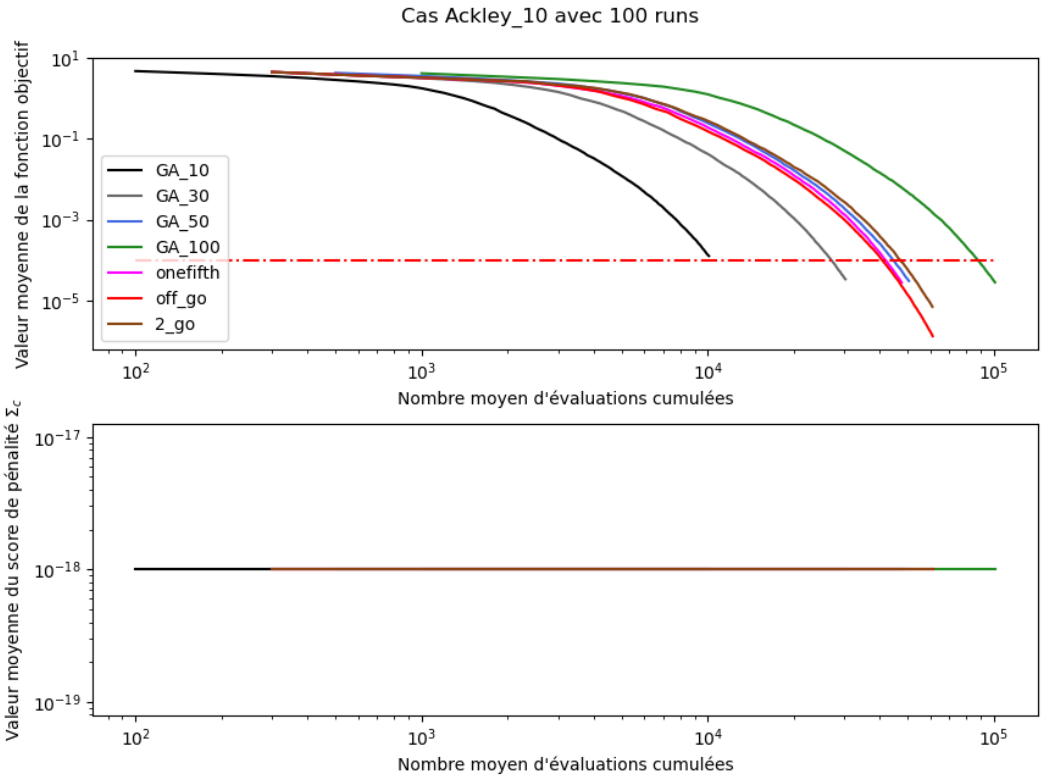
Tableau 22: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Ackley_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_10 (GA_10)	[2.25e-05,7.46e-04]	1.27e-04	9.17e-05	0.00e+00
GA_30 (GA_30)	[9.20e-06,8.22e-05]	3.36e-05	3.12e-05	0.00e+00
GA_50 (GA_50)	[9.68e-06,9.59e-05]	3.05e-05	2.83e-05	0.00e+00
GA_100 (GA_100)	[1.17e-05,5.83e-05]	2.81e-05	2.63e-05	0.00e+00
GA_OneFifth (onefifth)	[1.06e-05,6.26e-05]	2.81e-05	2.46e-05	0.00e+00
GA_Offspring_GO (off_go)	[8.18e-08,4.55e-06]	1.32e-06	1.15e-06	0.00e+00
GA_Both_GO (2_go)	[1.49e-06,1.91e-05]	7.01e-06	6.05e-06	0.00e+00

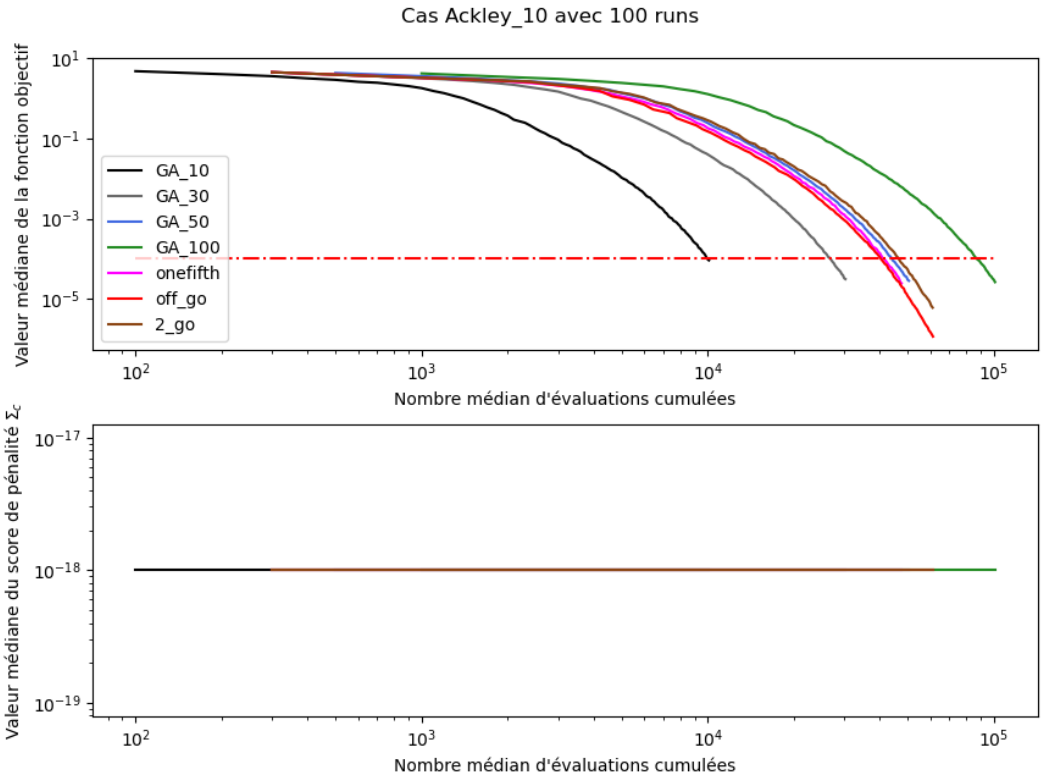
Tableau 23: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Ackley_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	GA_10	GA_30	GA_50	GA_100	onefifth	off_go	2_go
GA_10	1.00e+00	3.34e-10	4.58e-13	1.92e-14	1.85e-15	3.91e-88	7.63e-60
GA_30	3.34e-10	1.00e+00	1.00e+00	1.00e+00	1.00e+00	3.99e-39	3.47e-21
GA_50	4.58e-13	1.00e+00	1.00e+00	1.00e+00	1.00e+00	4.57e-34	1.66e-17
GA_100	1.92e-14	1.00e+00	1.00e+00	1.00e+00	1.00e+00	6.13e-32	5.48e-16
onefifth	1.85e-15	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.79e-30	5.93e-15
off_go	3.91e-88	3.99e-39	4.57e-34	6.13e-32	1.79e-30	1.00e+00	7.99e-03
2_go	7.63e-60	3.47e-21	1.66e-17	5.48e-16	5.93e-15	7.99e-03	1.00e+00

Tableau 24: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Ackley_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

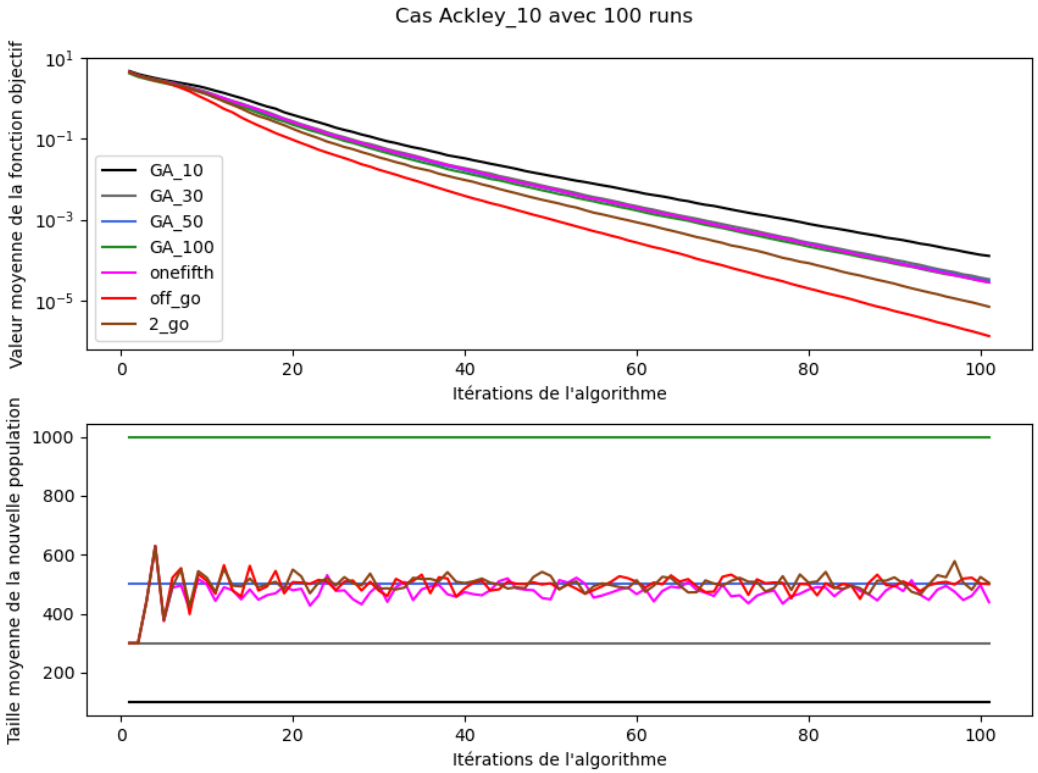


(a) Graphiques des résultats moyens.

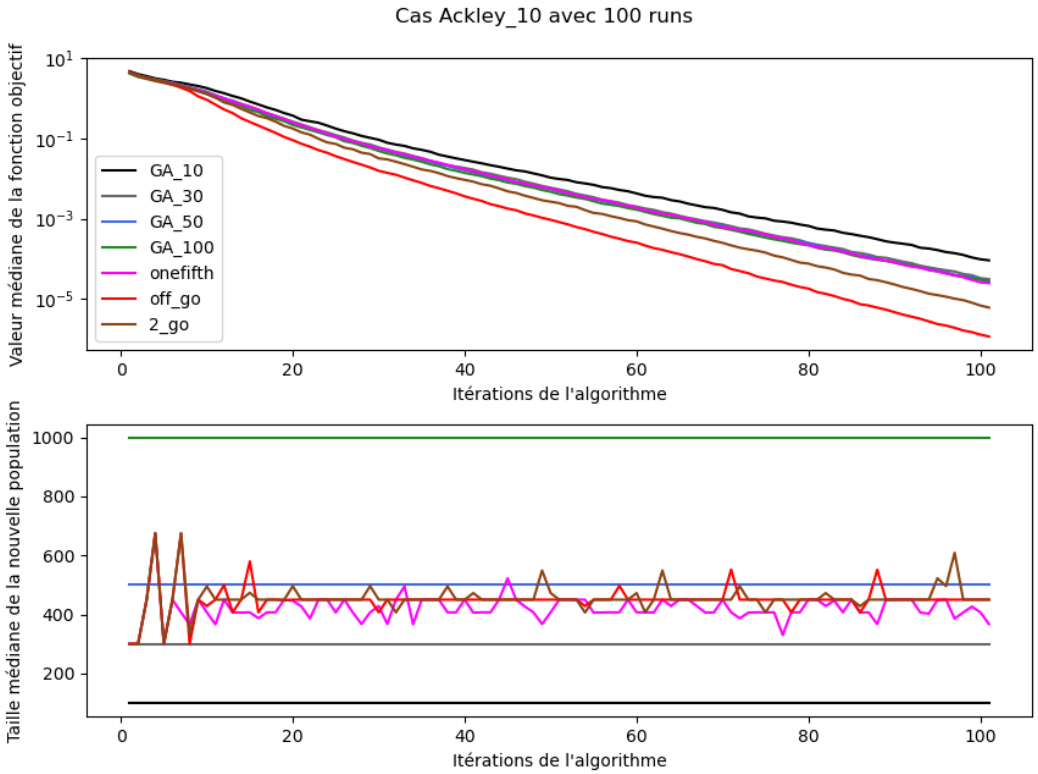


(b) Graphiques des résultats médians.

Figure 15: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Ackley_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 16: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Ackley_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test G2_plog_10

GA_10 (GA_10)	29.0%
GA_30 (GA_30)	53.0%
GA_50 (GA_50)	67.0%
GA_100 (GA_100)	68.0%
GA_OneFifth (onefifth)	65.0%
GA_Offspring_GO (off_go)	53.0%
GA_Both_GO (2_go)	73.0%

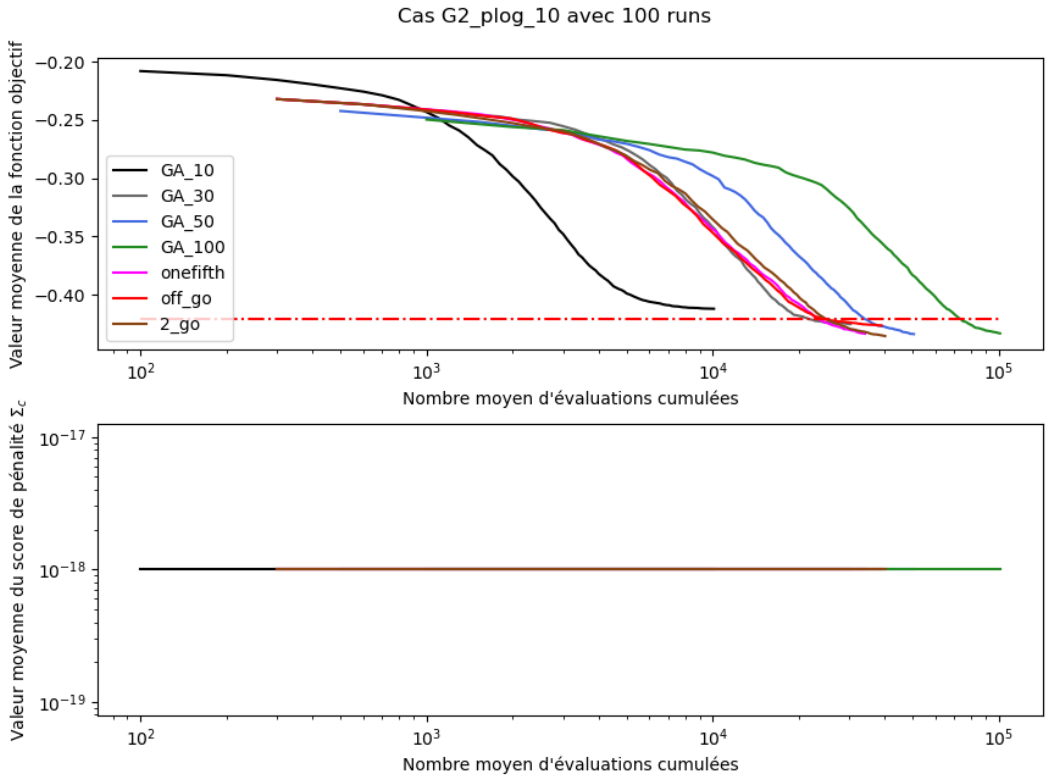
Tableau 25: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test G2_plog_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_10 (GA_10)	[2.31e-01,5.64e-01]	4.44e-01	4.51e-01	0.00e+00
GA_30 (GA_30)	[2.53e-01,5.13e-01]	4.26e-01	4.30e-01	0.00e+00
GA_50 (GA_50)	[2.96e-01,4.77e-01]	4.14e-01	4.21e-01	0.00e+00
GA_100 (GA_100)	[3.21e-01,4.86e-01]	4.15e-01	4.17e-01	0.00e+00
GA_OneFifth (onefifth)	[2.37e-01,4.96e-01]	4.15e-01	4.21e-01	0.00e+00
GA_Offspring_GO (off_go)	[3.43e-01,5.04e-01]	4.24e-01	4.29e-01	0.00e+00
GA_Both_GO (2_go)	[2.72e-01,4.79e-01]	4.12e-01	4.18e-01	0.00e+00

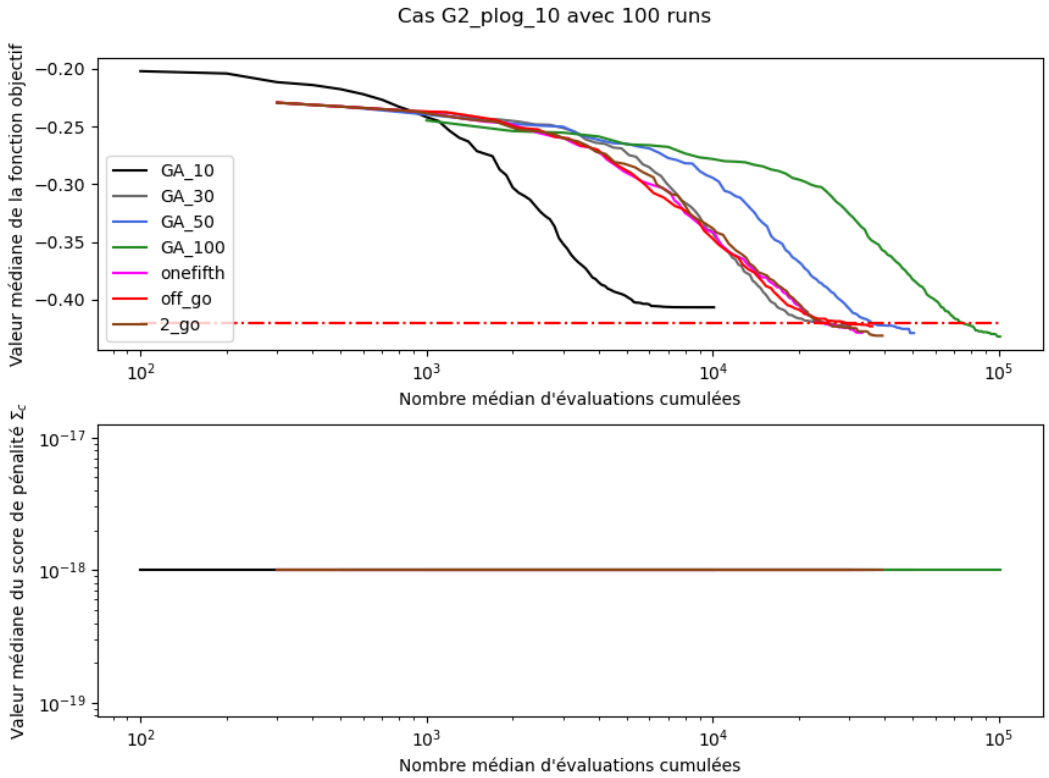
Tableau 26: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test G2_plog_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	GA_10	GA_30	GA_50	GA_100	onefifth	off_go	2_go
GA_10	1.00e+00	8.12e-03	1.56e-07	2.80e-08	7.03e-07	2.04e-03	2.35e-09
GA_30	8.12e-03	1.00e+00	5.38e-01	2.50e-01	1.00e+00	1.00e+00	7.81e-02
GA_50	1.56e-07	5.38e-01	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00
GA_100	2.80e-08	2.50e-01	1.00e+00	1.00e+00	1.00e+00	6.37e-01	1.00e+00
onefifth	7.03e-07	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00
off_go	2.04e-03	1.00e+00	1.00e+00	6.37e-01	1.00e+00	1.00e+00	2.25e-01
2_go	2.35e-09	7.81e-02	1.00e+00	1.00e+00	1.00e+00	2.25e-01	1.00e+00

Tableau 27: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test G2_plog_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

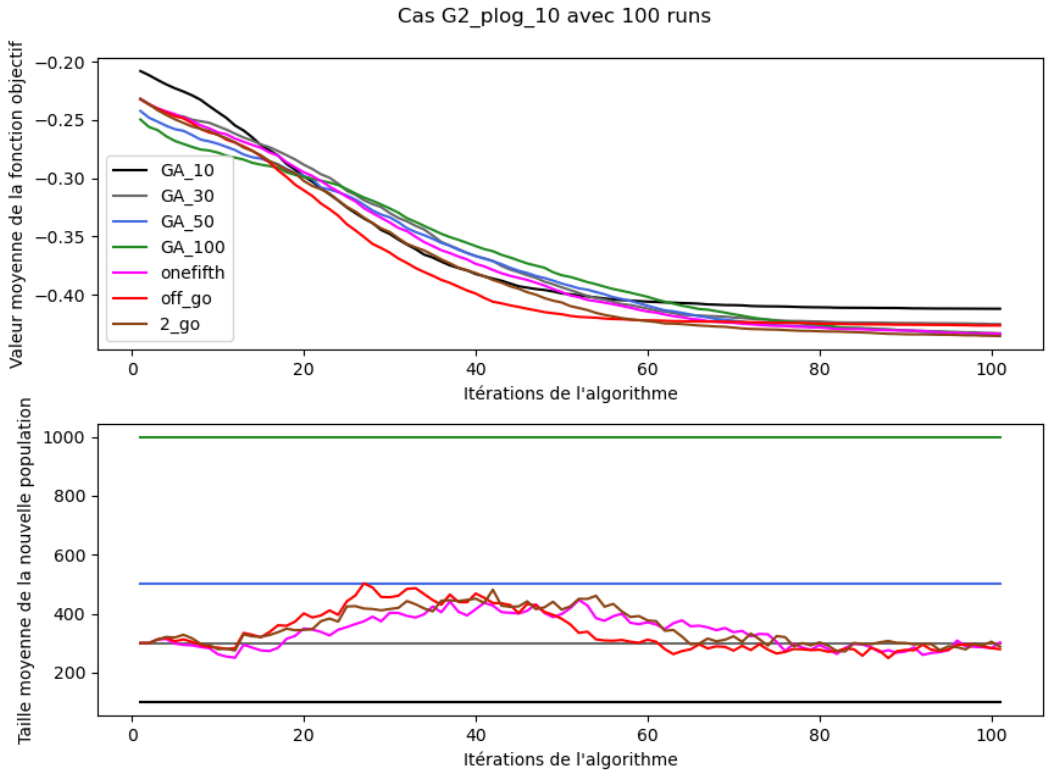


(a) Graphiques des résultats moyens.

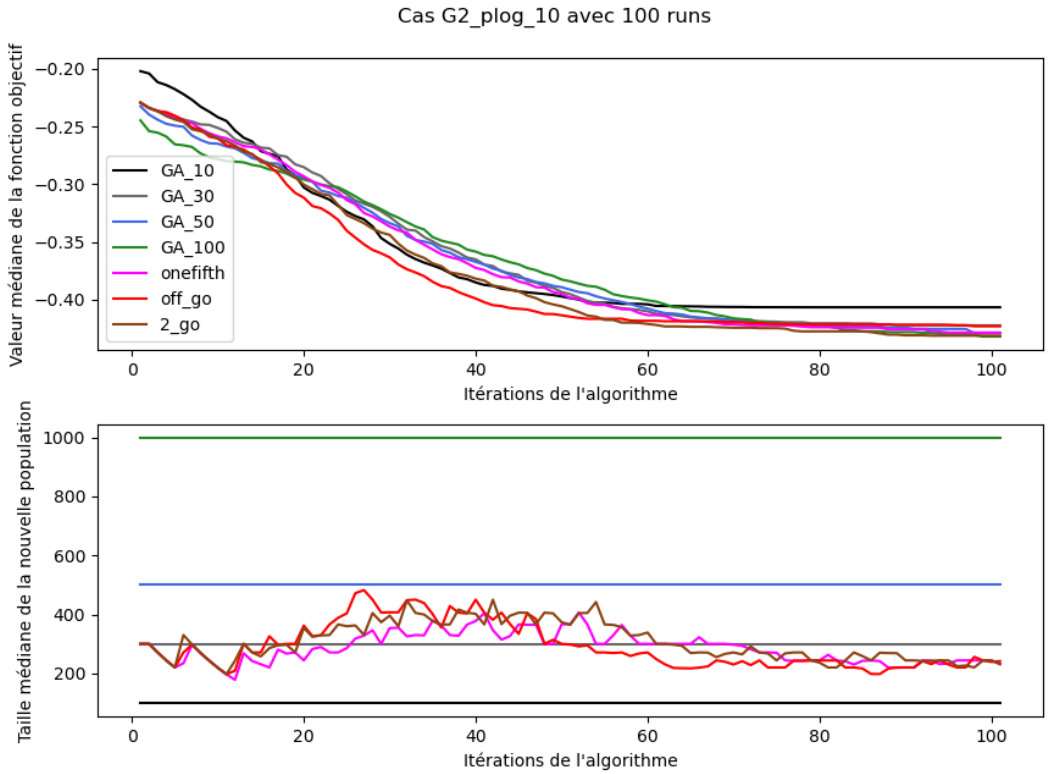


(b) Graphiques des résultats médians.

FIGURE 17: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test G2_plog_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 18: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test G2_plog_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test G7

GA_10 (GA_10)	78.0%
GA_30 (GA_30)	97.0%
GA_50 (GA_50)	99.0%
GA_100 (GA_100)	100.0%
GA_OneFifth (onefifth)	99.0%
GA_Offspring_GO (off_go)	99.0%
GA_Both_GO (2_go)	100.0%

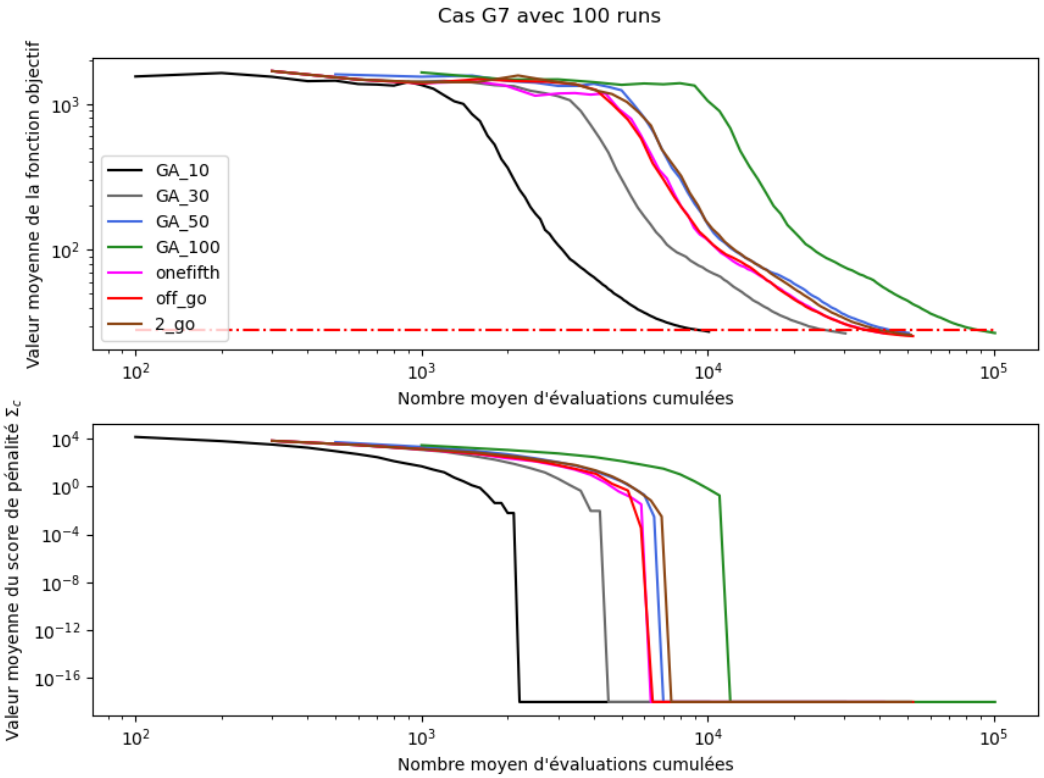
Tableau 28: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test G7.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_10 (GA_10)	[3.97e-02,5.56e-01]	1.25e-01	1.03e-01	0.00e+00
GA_30 (GA_30)	[4.46e-02,2.00e-01]	9.42e-02	9.16e-02	0.00e+00
GA_50 (GA_50)	[5.96e-02,1.71e-01]	1.01e-01	9.84e-02	0.00e+00
GA_100 (GA_100)	[3.64e-02,1.50e-01]	1.00e-01	1.02e-01	0.00e+00
GA_OneFifth (onefifth)	[5.94e-02,1.70e-01]	9.88e-02	9.72e-02	0.00e+00
GA_Offspring_GO (off_go)	[2.16e-02,1.65e-01]	4.79e-02	4.52e-02	0.00e+00
GA_Both_GO (2_go)	[3.20e-02,1.38e-01]	6.77e-02	6.62e-02	0.00e+00

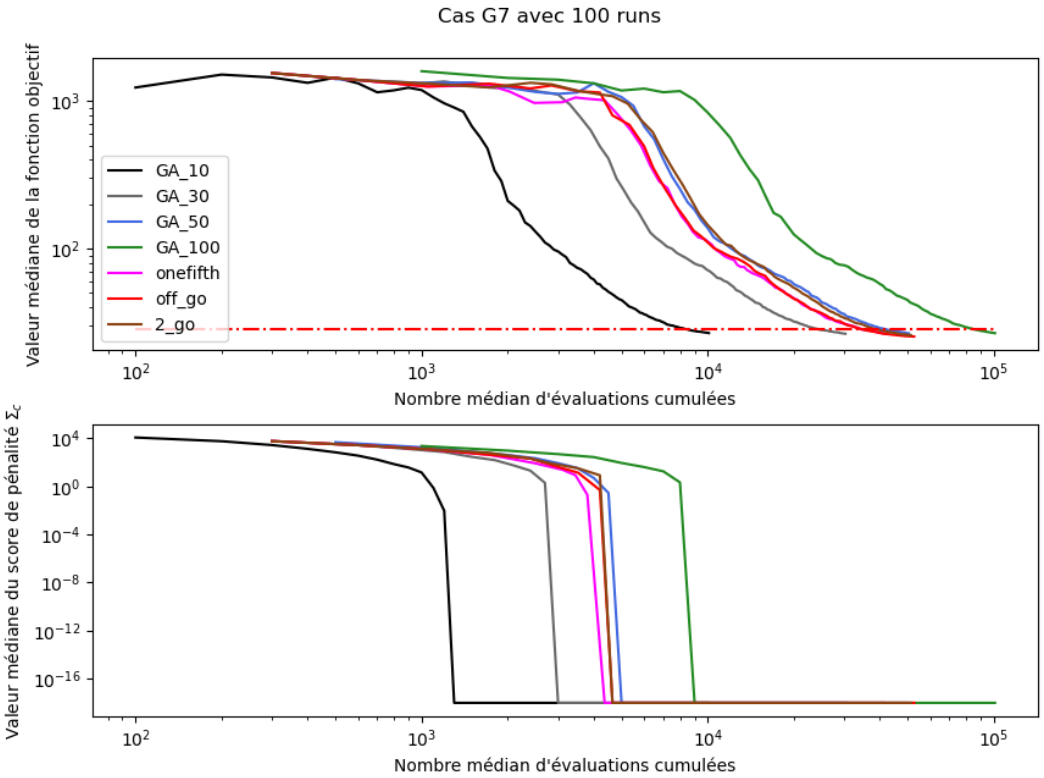
Tableau 29: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test G7. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	GA_10	GA_30	GA_50	GA_100	onefifth	off_go	2_go
GA_10	1.00e+00	7.31e-02	1.00e+00	1.00e+00	1.00e+00	9.20e-38	2.05e-18
GA_30	7.31e-02	1.00e+00	6.42e-01	3.23e-01	1.00e+00	6.52e-23	1.44e-08
GA_50	1.00e+00	6.42e-01	1.00e+00	1.00e+00	1.00e+00	1.50e-33	1.66e-15
GA_100	1.00e+00	3.23e-01	1.00e+00	1.00e+00	1.00e+00	5.76e-35	1.79e-16
onefifth	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.26e-29	7.18e-13
off_go	9.20e-38	6.52e-23	1.50e-33	5.76e-35	1.26e-29	1.00e+00	1.41e-03
2_go	2.05e-18	1.44e-08	1.66e-15	1.79e-16	7.18e-13	1.41e-03	1.00e+00

Tableau 30: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test G7. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

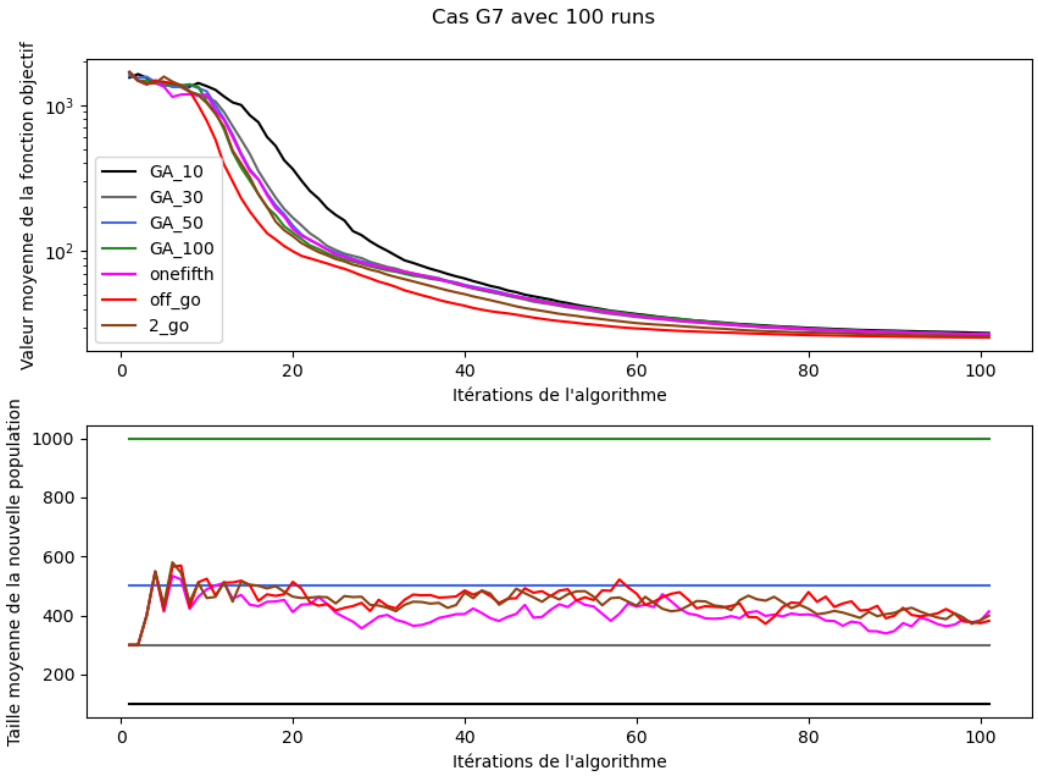


(a) Graphiques des résultats moyens.

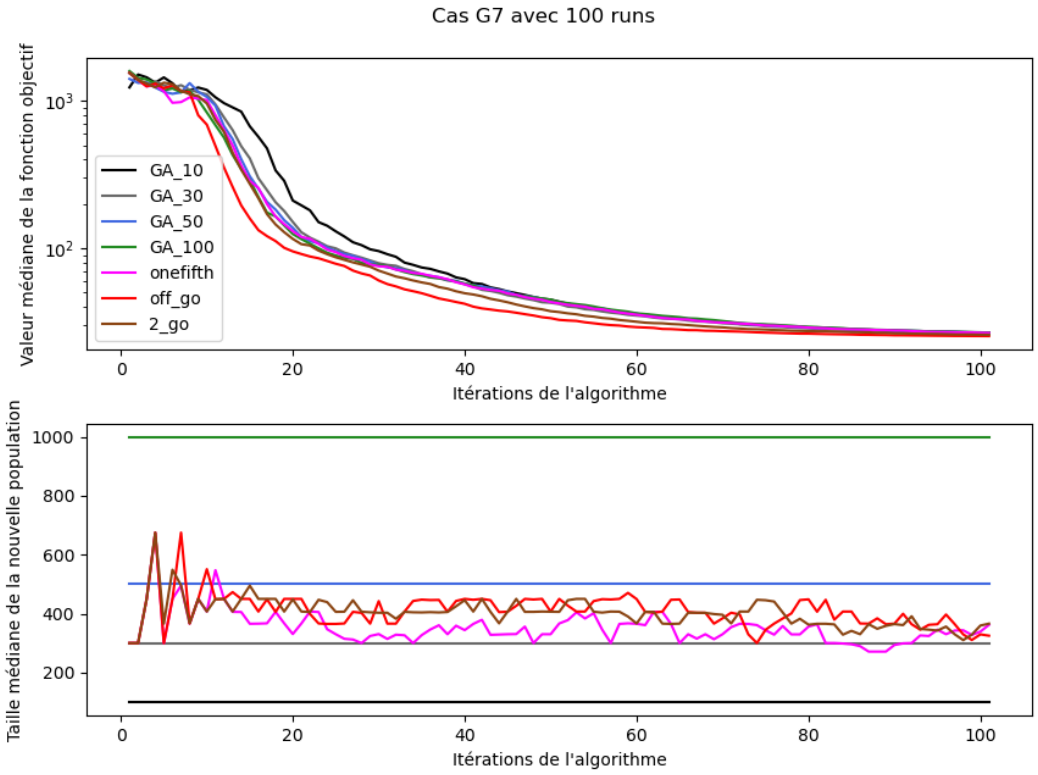


(b) Graphiques des résultats médians.

FIGURE 19: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test G7. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 20: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test G7. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test G10

GA_10 (GA_10)	48.0%
GA_30 (GA_30)	55.0%
GA_50 (GA_50)	36.0%
GA_100 (GA_100)	12.0%
GA_OneFifth (onefifth)	57.0%
GA_Offspring_GO (off_go)	85.0%
GA_Both_GO (2_go)	75.0%

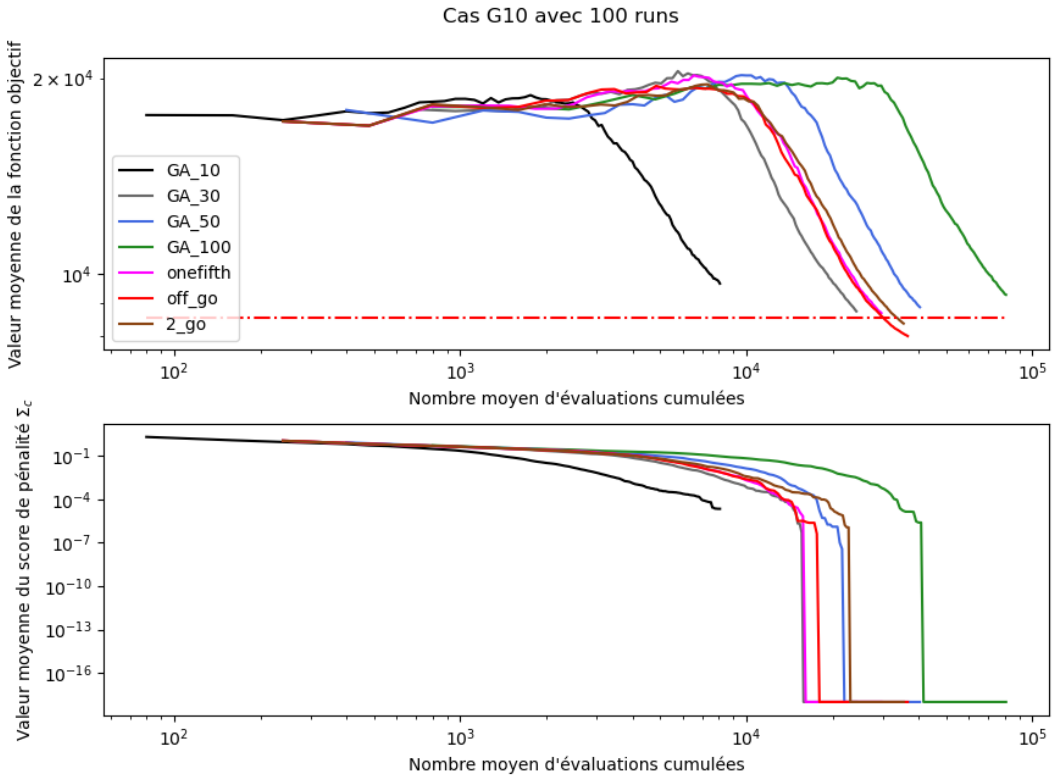
Tableau 31: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test G10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_10 (GA_10)	[4.96e-02,2.33e+00]	3.69e-01	2.19e-01	0.00e+00
GA_30 (GA_30)	[1.04e-01,7.07e-01]	2.41e-01	2.05e-01	0.00e+00
GA_50 (GA_50)	[1.02e-01,4.97e-01]	2.59e-01	2.47e-01	0.00e+00
GA_100 (GA_100)	[1.55e-01,5.56e-01]	3.16e-01	3.03e-01	0.00e+00
GA_OneFifth (onefifth)	[7.29e-02,7.64e-01]	2.33e-01	2.06e-01	0.00e+00
GA_Offspring_GO (off_go)	[4.96e-02,4.91e-01]	1.36e-01	1.14e-01	0.00e+00
GA_Both_GO (2_go)	[8.13e-02,1.03e+00]	1.88e-01	1.51e-01	0.00e+00

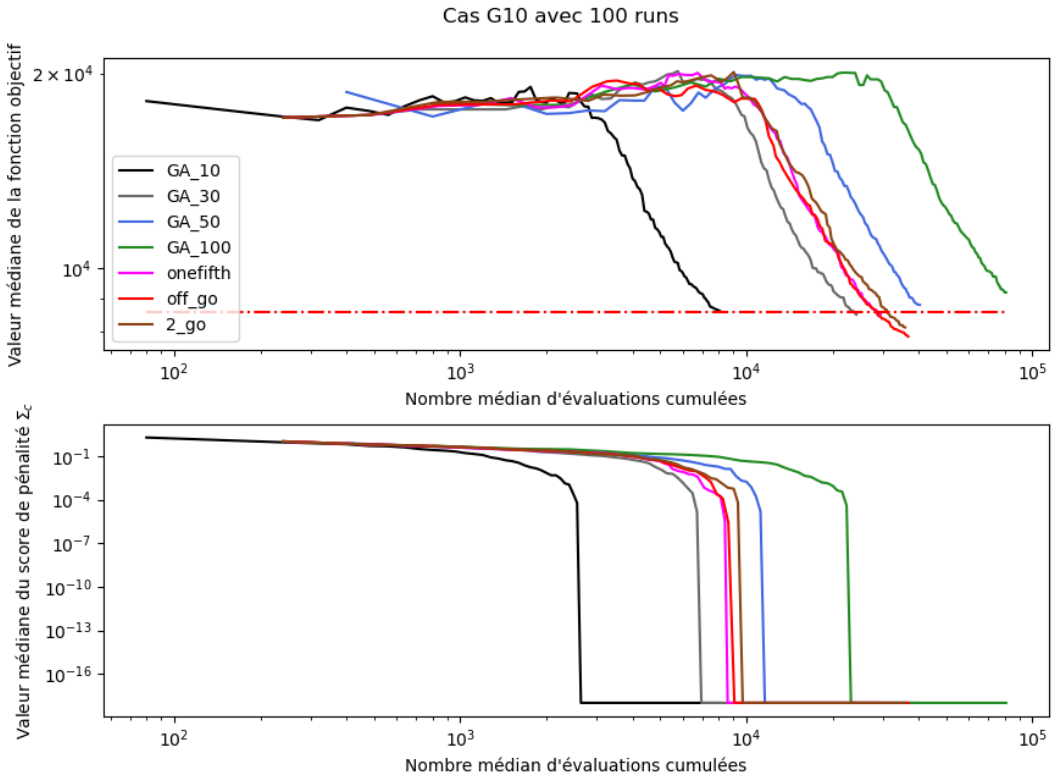
Tableau 32: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test G10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	GA_10	GA_30	GA_50	GA_100	onefifth	off_go	2_go
GA_10	1.00e+00	1.00e+00	1.00e+00	6.76e-04	1.00e+00	7.64e-16	5.71e-06
GA_30	1.00e+00	1.00e+00	6.29e-01	1.49e-06	1.00e+00	1.34e-11	1.93e-03
GA_50	1.00e+00	6.29e-01	1.00e+00	2.71e-02	1.72e-01	1.62e-19	2.50e-08
GA_100	6.76e-04	1.49e-06	2.71e-02	1.00e+00	9.56e-08	5.61e-35	2.97e-19
onefifth	1.00e+00	1.00e+00	1.72e-01	9.56e-08	1.00e+00	3.85e-10	1.24e-02
off_go	7.64e-16	1.34e-11	1.62e-19	5.61e-35	3.85e-10	1.00e+00	2.16e-02
2_go	5.71e-06	1.93e-03	2.50e-08	2.97e-19	1.24e-02	2.16e-02	1.00e+00

Tableau 33: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test G10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

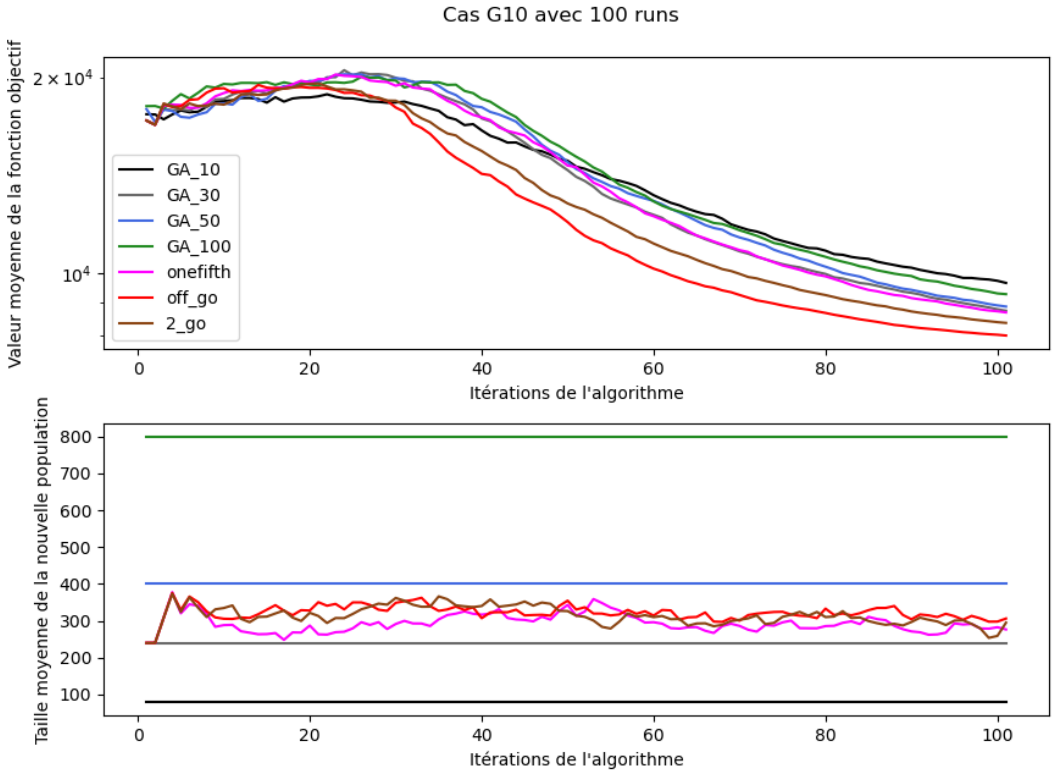


(a) Graphiques des résultats moyens.

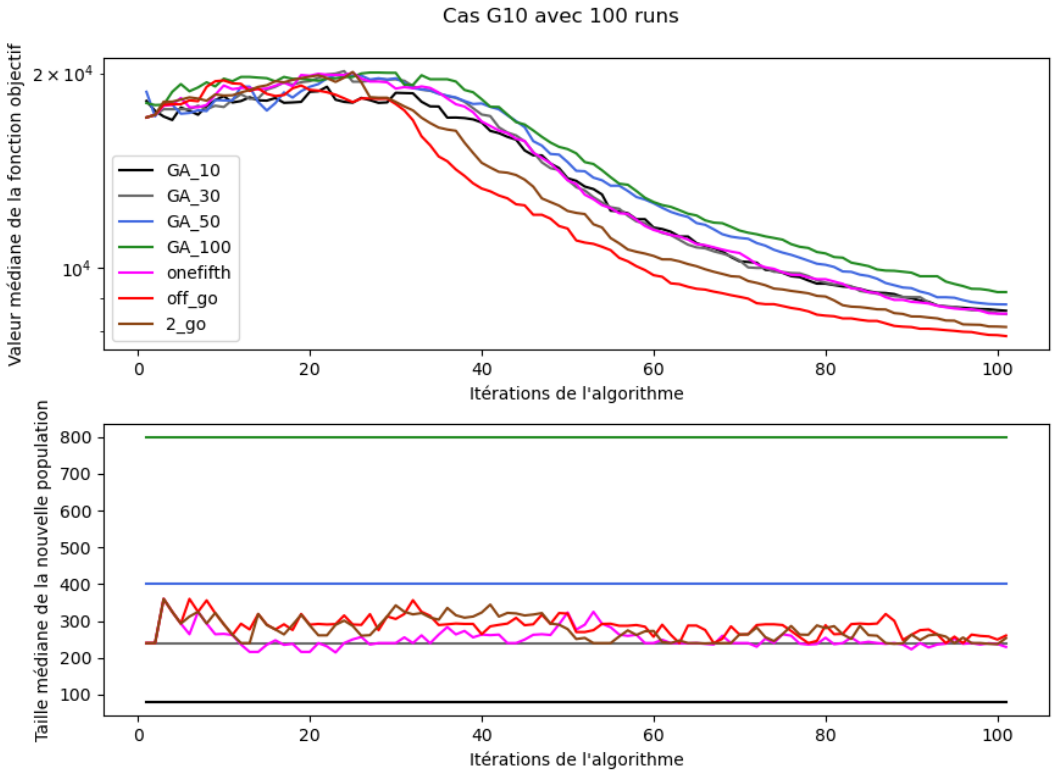


(b) Graphiques des résultats médians.

FIGURE 21: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test G10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 22: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test G10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test Rastrigin_10

GA_10 (GA_10)	13.0%
GA_30 (GA_30)	50.0%
GA_50 (GA_50)	52.0%
GA_100 (GA_100)	27.0%
GA_OneFifth (onefifth)	37.0%
GA_Offspring_GO (off_go)	83.0%
GA_Both_GO (2_go)	77.0%

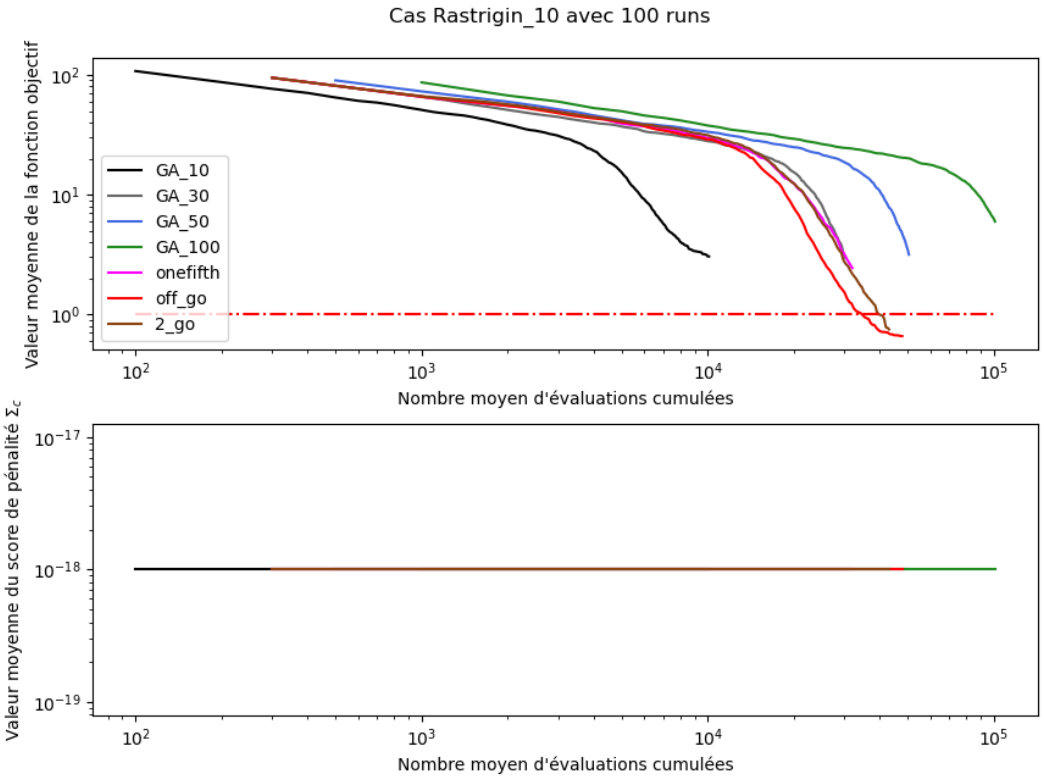
Tableau 34: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Rastrigin_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_10 (GA_10)	[1.89e-03,1.36e+01]	3.04e+00	2.99e+00	0.00e+00
GA_30 (GA_30)	[2.71e-04,1.99e+01]	2.97e+00	1.00e+00	0.00e+00
GA_50 (GA_50)	[5.18e-04,1.91e+01]	3.16e+00	8.83e-01	0.00e+00
GA_100 (GA_100)	[5.20e-02,1.90e+01]	5.96e+00	3.85e+00	0.00e+00
GA_OneFifth (onefifth)	[2.11e-04,1.78e+01]	2.44e+00	1.10e+00	0.00e+00
GA_Offspring_GO (off_go)	[3.60e-08,5.98e+00]	6.57e-01	9.65e-05	0.00e+00
GA_Both_GO (2_go)	[6.33e-06,1.25e+01]	7.48e-01	9.88e-02	0.00e+00

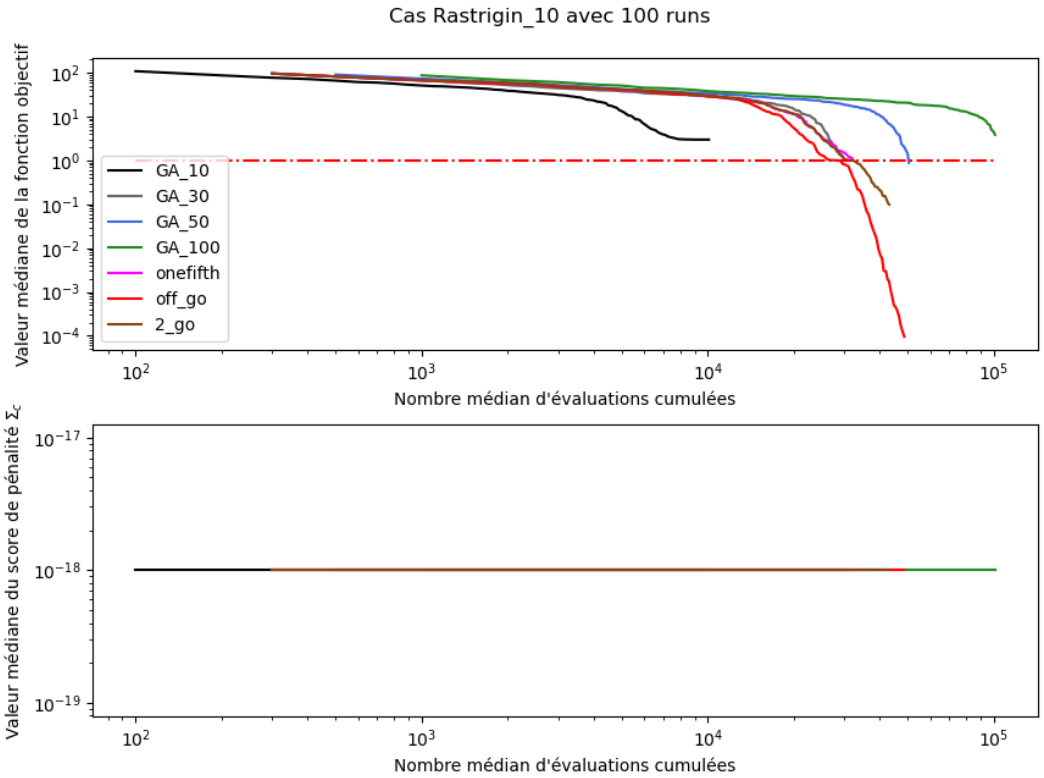
Tableau 35: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Rastrigin_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	GA_10	GA_30	GA_50	GA_100	onefifth	off_go	2_go
GA_10	1.00e+00	1.13e-04	3.46e-04	1.00e+00	1.25e-03	1.32e-26	7.54e-19
GA_30	1.13e-04	1.00e+00	1.00e+00	2.43e-05	1.00e+00	3.14e-09	6.95e-05
GA_50	3.46e-04	1.00e+00	1.00e+00	7.99e-05	1.00e+00	6.30e-10	2.10e-05
GA_100	1.00e+00	2.43e-05	7.99e-05	1.00e+00	3.17e-04	3.95e-28	3.88e-20
onefifth	1.25e-03	1.00e+00	1.00e+00	3.17e-04	1.00e+00	8.19e-11	4.52e-06
off_go	1.32e-26	3.14e-09	6.30e-10	3.95e-28	8.19e-11	1.00e+00	1.00e+00
2_go	7.54e-19	6.95e-05	2.10e-05	3.88e-20	4.52e-06	1.00e+00	1.00e+00

Tableau 36: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Rastrigin_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

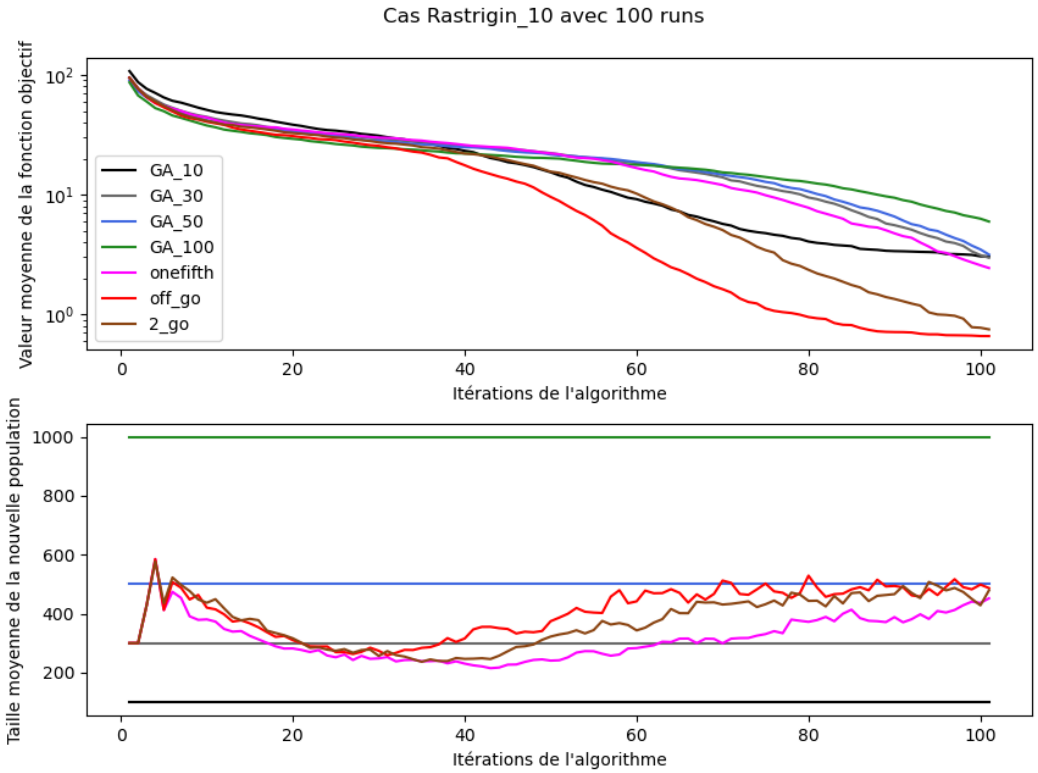


(a) Graphiques des résultats moyens.

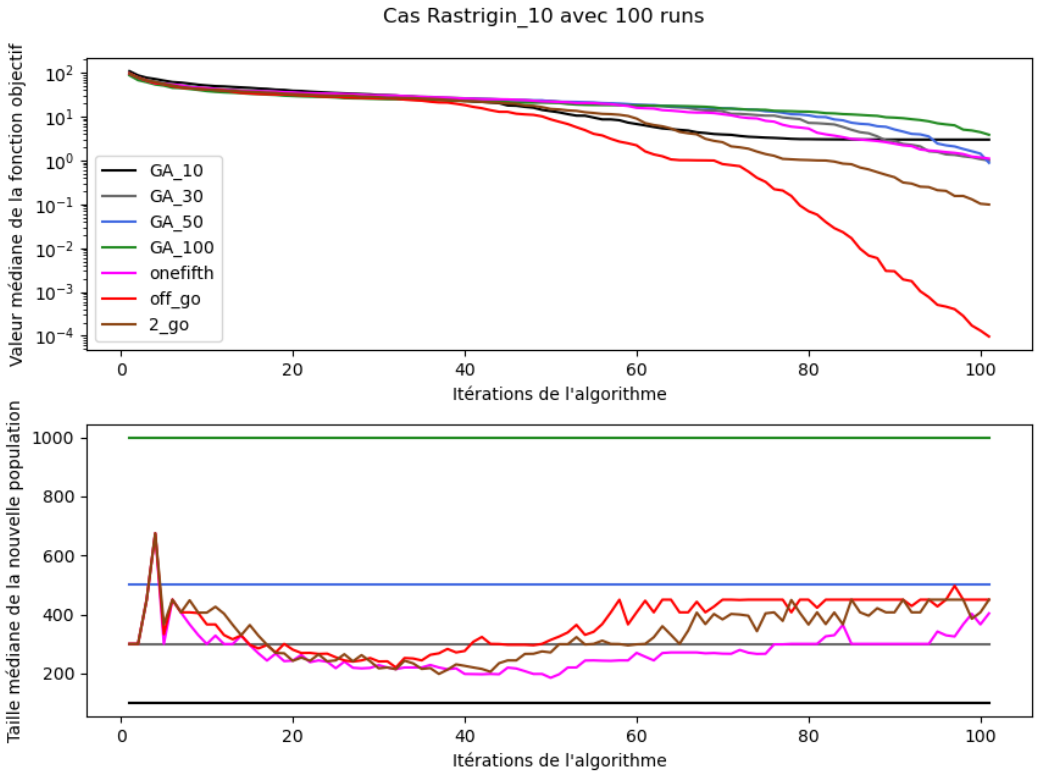


(b) Graphiques des résultats médians.

FIGURE 23: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Rastrigin_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 24: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Rastrigin_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test Rosenbrock_10

GA_10 (GA_10)	15.0%
GA_30 (GA_30)	44.0%
GA_50 (GA_50)	62.0%
GA_100 (GA_100)	66.0%
GA_OneFifth (onefifth)	72.0%
GA_Offspring_GO (off_go)	86.0%
GA_Both_GO (2_go)	84.0%

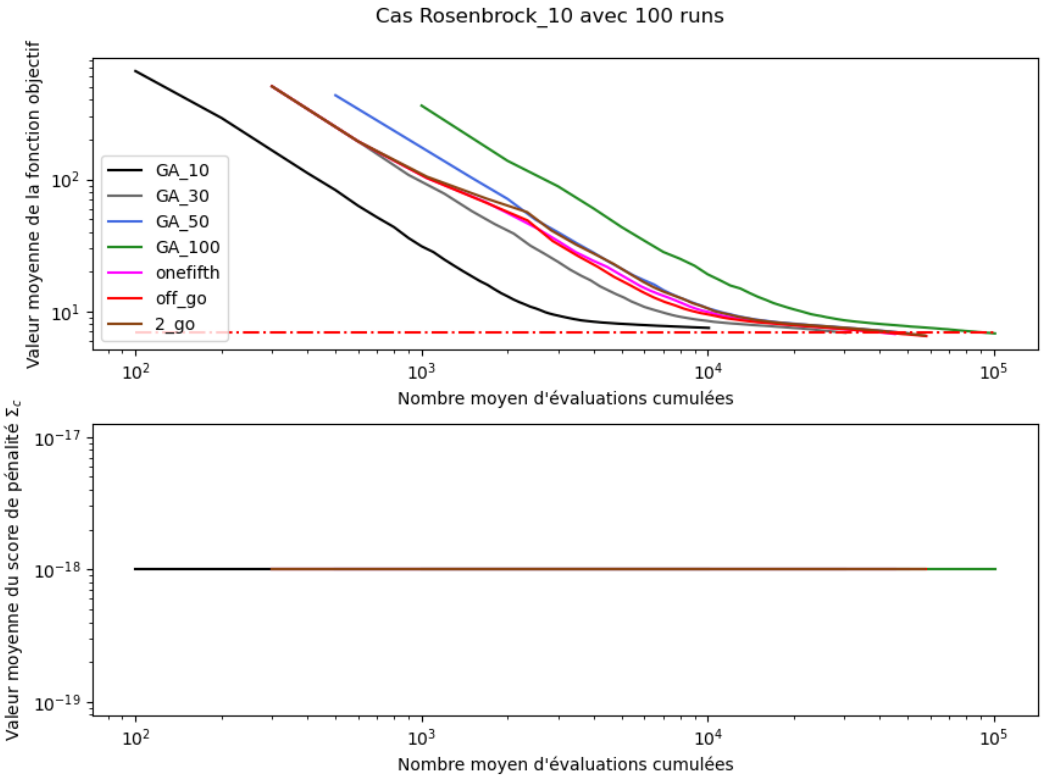
Tableau 37: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Rosenbrock_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_10 (GA_10)	[5.03e+00,9.60e+00]	7.48e+00	7.46e+00	0.00e+00
GA_30 (GA_30)	[6.07e+00,7.57e+00]	6.88e+00	6.93e+00	0.00e+00
GA_50 (GA_50)	[6.17e+00,7.29e+00]	6.80e+00	6.84e+00	0.00e+00
GA_100 (GA_100)	[6.17e+00,7.26e+00]	6.79e+00	6.81e+00	0.00e+00
GA_OneFifth (onefifth)	[6.02e+00,7.31e+00]	6.73e+00	6.74e+00	0.00e+00
GA_Offspring_GO (off_go)	[5.26e+00,7.24e+00]	6.49e+00	6.52e+00	0.00e+00
GA_Both_GO (2_go)	[5.69e+00,7.14e+00]	6.56e+00	6.58e+00	0.00e+00

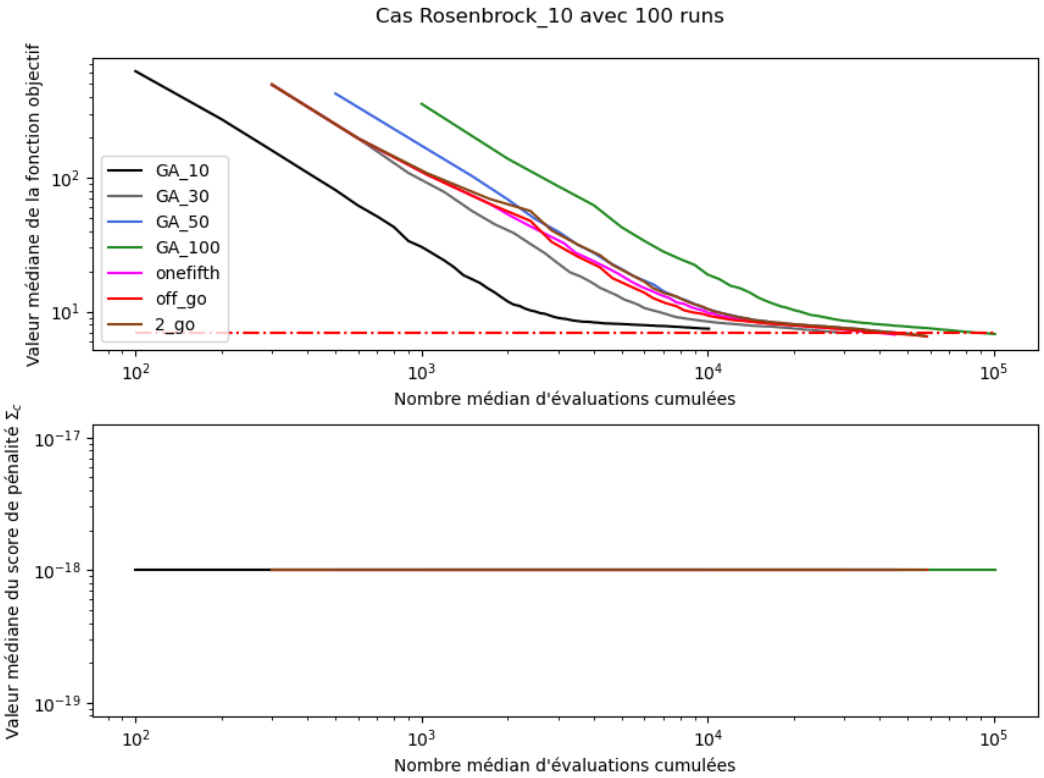
Tableau 38: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Rosenbrock_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	GA_10	GA_30	GA_50	GA_100	onefifth	off_go	2_go
GA_10	1.00e+00	8.52e-08	1.01e-12	2.55e-13	1.16e-18	2.50e-38	2.00e-32
GA_30	8.52e-08	1.00e+00	1.00e+00	1.00e+00	2.25e-02	6.27e-12	1.00e-08
GA_50	1.01e-12	1.00e+00	1.00e+00	1.00e+00	1.00e+00	3.56e-07	1.02e-04
GA_100	2.55e-13	1.00e+00	1.00e+00	1.00e+00	1.00e+00	9.87e-07	2.34e-04
onefifth	1.16e-18	2.25e-02	1.00e+00	1.00e+00	1.00e+00	1.20e-03	6.56e-02
off_go	2.50e-38	6.27e-12	3.56e-07	9.87e-07	1.20e-03	1.00e+00	1.00e+00
2_go	2.00e-32	1.00e-08	1.02e-04	2.34e-04	6.56e-02	1.00e+00	1.00e+00

Tableau 39: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Rosenbrock_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

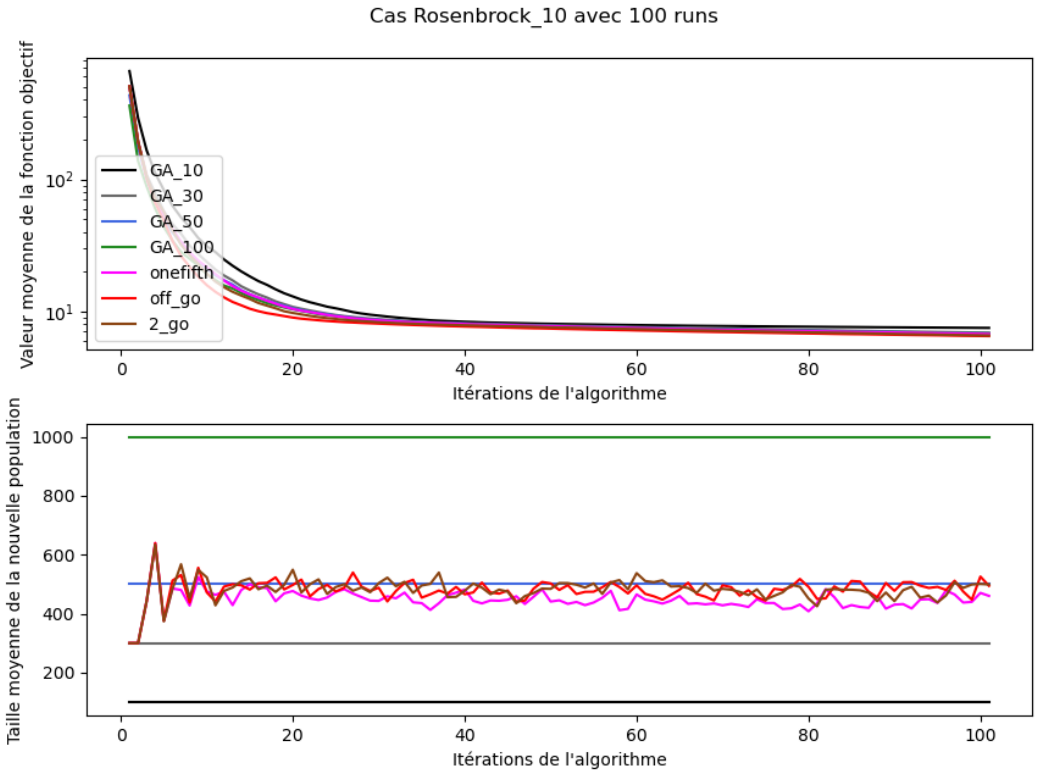


(a) Graphiques des résultats moyens.

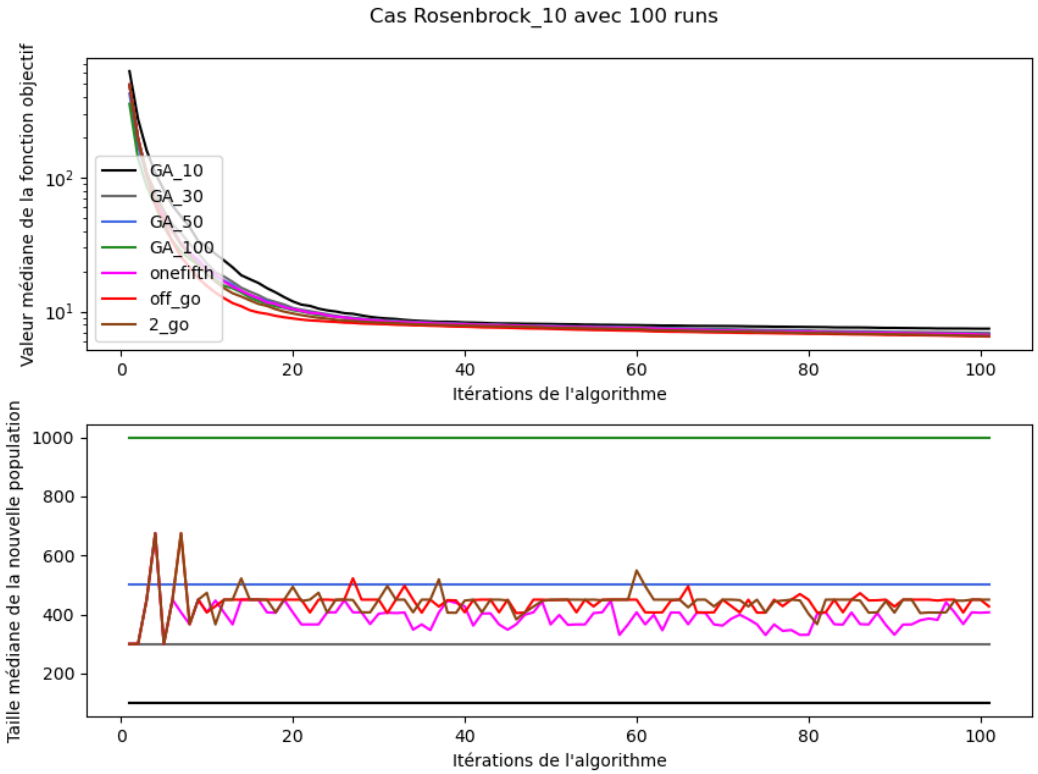


(b) Graphiques des résultats médians.

FIGURE 25: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Rosenbrock_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 26: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Rosenbrock_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.

Cas-test Schwefel_10

GA_10 (GA_10)	82.0%
GA_30 (GA_30)	44.0%
GA_50 (GA_50)	60.0%
GA_100 (GA_100)	88.0%
GA_OneFifth (onefifth)	47.0%
GA_Offspring_GO (off_go)	61.0%
GA_Both_GO (2_go)	57.0%

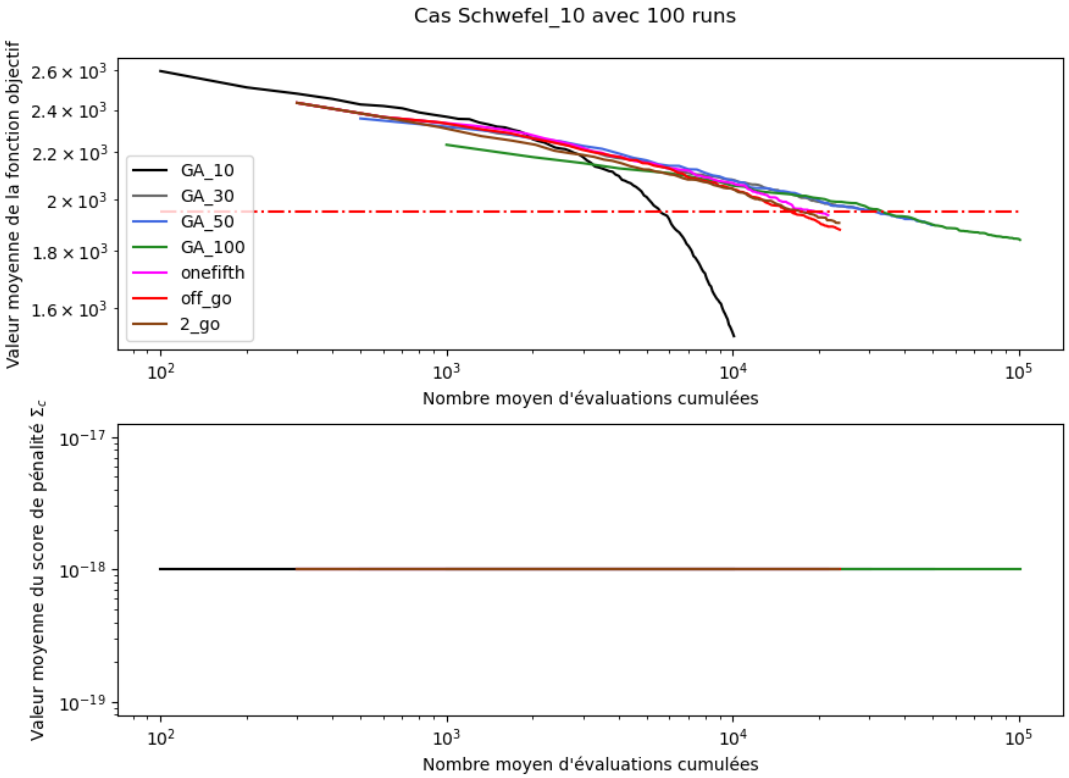
Tableau 40: Présentation des taux d'exécutions réussies pour chaque algorithme considéré sur le cas-test Schwefel_10.

versions	[min,max]	moyenne	médiane	médiane (Σ_c)
GA_10 (GA_10)	[4.24e+02,2.17e+03]	1.51e+03	1.59e+03	0.00e+00
GA_30 (GA_30)	[1.57e+03,2.17e+03]	1.95e+03	1.98e+03	0.00e+00
GA_50 (GA_50)	[1.47e+03,2.20e+03]	1.90e+03	1.92e+03	0.00e+00
GA_100 (GA_100)	[1.48e+03,2.08e+03]	1.84e+03	1.87e+03	0.00e+00
GA_OneFifth (onefifth)	[1.17e+03,2.30e+03]	1.94e+03	1.96e+03	0.00e+00
GA_Offspring_GO (off_go)	[1.10e+03,2.19e+03]	1.88e+03	1.91e+03	0.00e+00
GA_Both_GO (2_go)	[1.33e+03,2.21e+03]	1.91e+03	1.94e+03	0.00e+00

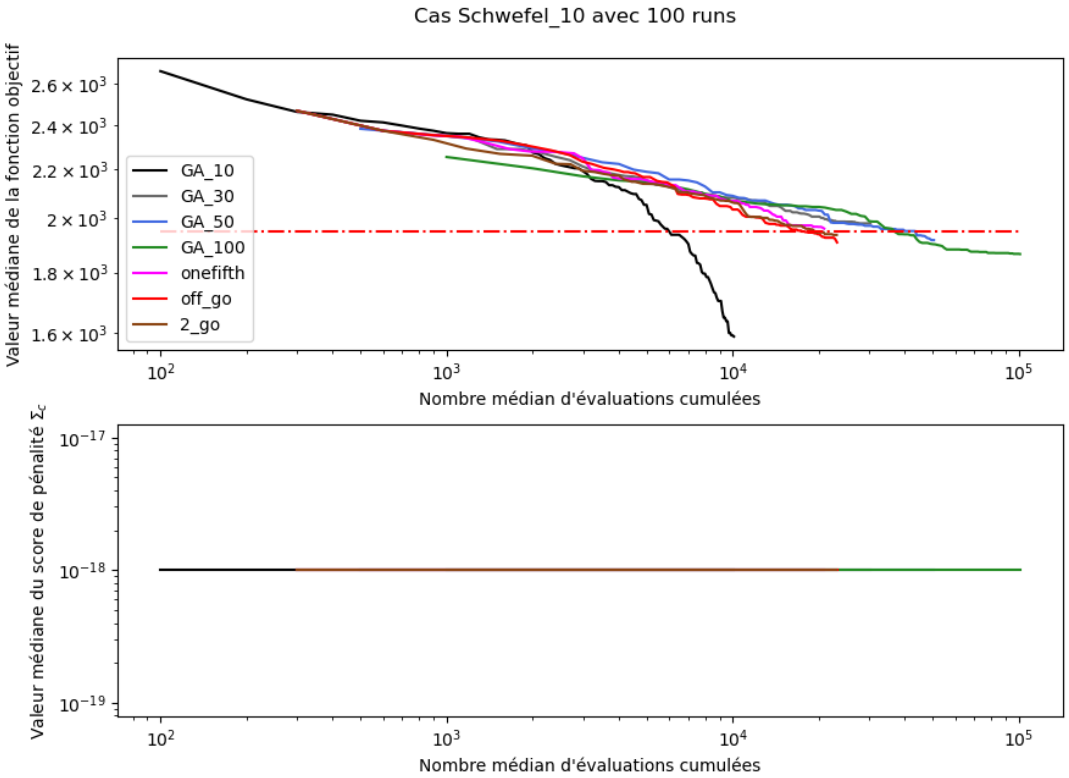
Tableau 41: Présentation des résultats statistiques de l'erreur relative pour les solutions finales proposées par les runs des différents algorithmes pour le cas-test Schwefel_10. La médiane du score de pénalité Σ_c est aussi visible pour avoir une information sur la tendance centrale du respect des contraintes. En ce qui concerne les médianes des erreurs relatives, si une case est significativement plus petite que les autres, selon le test statistique de Dunn [27], elle sera colorée en vert. Si plusieurs cases sont colorées, cela signifie que les médianes impliquées sont significativement plus petites que les autres et qu'elles ne sont pas significativement différentes entre elles.

	GA_10	GA_30	GA_50	GA_100	onefifth	off_go	2_go
GA_10	1.00e+00	5.78e-19	2.45e-10	5.37e-03	7.43e-17	2.14e-08	2.14e-11
GA_30	5.78e-19	1.00e+00	3.05e-01	5.30e-07	1.00e+00	3.78e-02	7.50e-01
GA_50	2.45e-10	3.05e-01	1.00e+00	3.70e-02	1.00e+00	1.00e+00	1.00e+00
GA_100	5.37e-03	5.30e-07	3.70e-02	1.00e+00	9.97e-06	3.00e-01	1.09e-02
onefifth	7.43e-17	1.00e+00	1.00e+00	9.97e-06	1.00e+00	2.04e-01	1.00e+00
off_go	2.14e-08	3.78e-02	1.00e+00	3.00e-01	2.04e-01	1.00e+00	1.00e+00
2_go	2.14e-11	7.50e-01	1.00e+00	1.09e-02	1.00e+00	1.00e+00	1.00e+00

Tableau 42: Présentation du résultat du test statistique de Dunn concernant les médianes des erreurs relatives pour le cas-test Schwefel_10. Si un couple d'algorithme présente une valeur inférieure ou égale à 0.05, cela signifiera que les médianes des erreurs relatives de ces deux algorithmes sont significativement différentes selon le test de Dunn [27]. Si c'est le cas, la case concernée sera coloriée en vert.

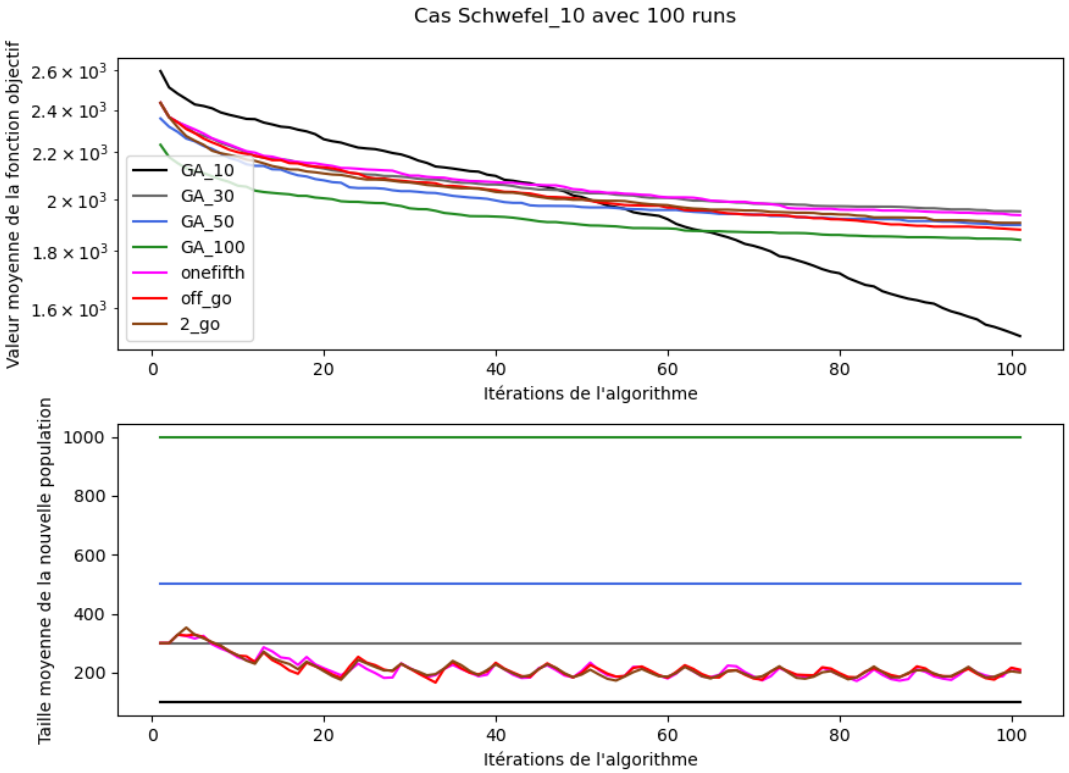


(a) Graphiques des résultats moyens.

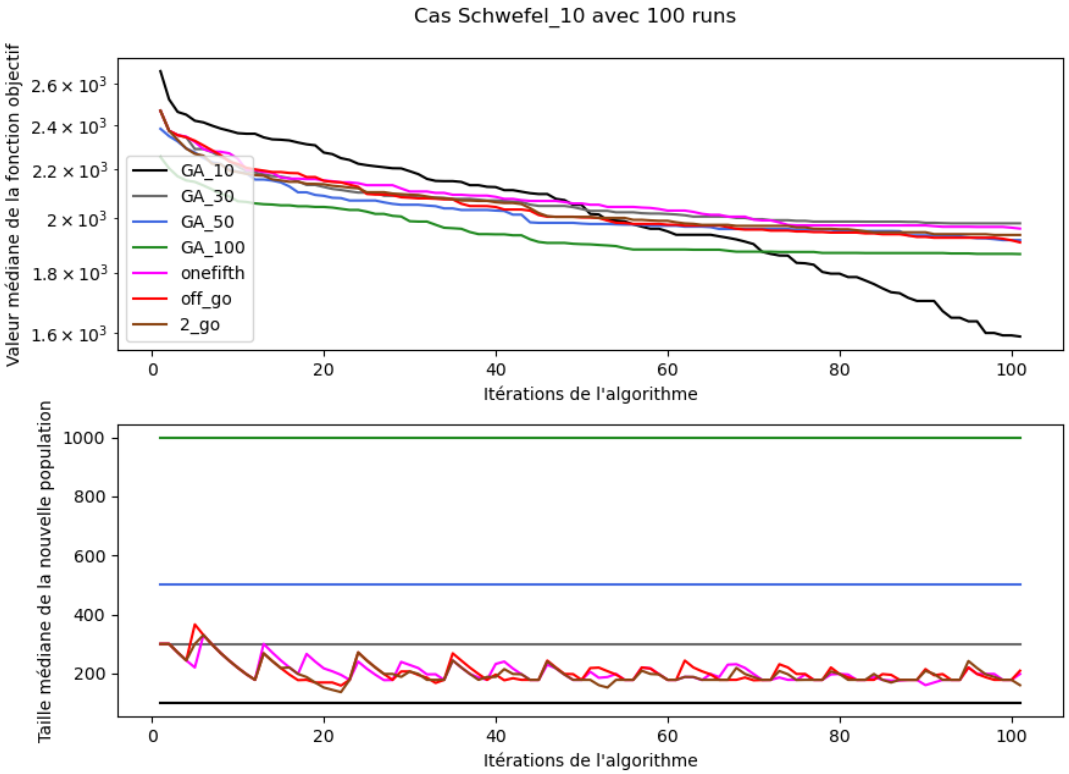


(b) Graphiques des résultats médians.

Figure 27: Évolution de la fonction objectif et du score de pénalité, tous les deux par rapport au nombre d'évaluations effectuées pour le cas-test Schwefel_10. Pour le graphique supérieur, le trait rouge horizontal représente la borne supérieure de l'erreur absolue b_{inf} définie dans (5.7). Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisés la médiane.



(a) Graphiques des résultats moyens.



(b) Graphiques des résultats médians.

Figure 28: Évolution de la fonction objectif et de la taille de la population par rapport au nombre d'itérations effectuées pour le cas-test Schwefel_10. Pour les graphiques (a), nous avons représenté la moyenne des différents runs et pour ceux de (b), nous avons utilisé la médiane.